VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Bezpečnostní model architektury .NET

Security Model for .NET Architecture

2009 Milada Píšová

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 7. května 2009 .....

Ráda bych na tomto místě poděkovala všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Tato práce se zaměřuje na prostředí platformy .NET a zajištění bezpečnosti s ohledem na spouštění cizího kódu. Popisuje, jak je tato platforma zabezpečena a jak lze s tímto zabezpečením pracovat.

Dále se zabývá způsoby testování zdrojových kódů studentů a nástroji použitelnými pro tento účel. Také navrhuje modul použitelný pro testování takovýchto zdrojových kódů. Součástí je i způsob zabezpečení tohoto modulu.

Klíčová slova: bezpečnost, .NET, diplomová práce, NUnit, MAUS, unit test

Abstract

This master thesis is about security in .NET platform and explores how to execute third-party code. It describe how is .NET secured and how to work with this security system.

Also this thesis describes possibilities for testing of student's source code and tools for this purpose. It designs module which will be used for testing these source codes. It also includes method of security this module.

Keywords: security, .NET, master thesis, NUnit, MAUS, unit test

Seznam použitých zkratek a symbolů

UML	–	Unified Modeling Language
SQL	–	Structured Query Language
CLR	–	Common Language Runtime
CAS	–	Code Access Security
URL	–	Uniform Resource Locator
JIT	–	Just-In-Time
MSIL	–	Microsoft Intermediate Language
CIL	–	Common Intermediate Language
GAC	–	Global Assembly Cache
ERD	–	Entity Relationship Diagram
XML	–	eXtensible Markup Language

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

S přibývajícím počtem studentů katedry informatiky na Vysoké škole báňské - Technické univerzitě v Ostravě je stále těžší a časově náročnější opravovat projekty v jednotlivých předmětech, hlavně pokud se jedná o projekty v programování, jelikož jedno zadání může mít spoustu správných řešení. Zjednodušením a hlavně urychlením této práce je převedení tohoto úkolu z vyučujícího na počítač. Toto řešení nabízí i spoustu dalších pozitiv, kterými se může stát například více menších úkolů pro studenty, které budou opraveny téměř okamžitě.

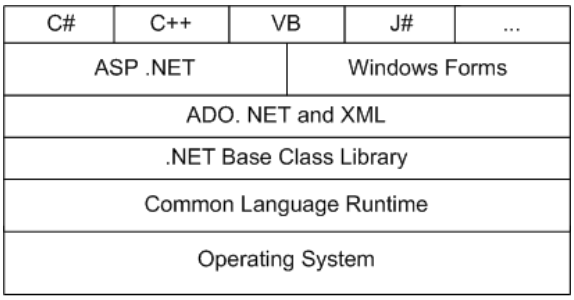
Na začátku je stručně popsána platforma .NET a dále pak už obsáhleji možnosti zabezpečení této platformy. Kapitola 3 se zaměřuje hlavně na zabezpečení spouštění cizího kódu. Následující dvě kapitoly dávají přehled, jakým způsobem lze testovat projekty studentů a které nástroje lze pro tyto účely použít.

Kapitola 6 se zabývá vývojem systému pro testování zdrojových kódů studentů v prostředí platformy .NET a jejích programovacích jazyků, speciálně se pak zaměřuje na jazyk C#. Více o vývoji tohoto systému popisují podkapitoly 6.1, 6.2 a 6.3.

2 .NET

.NET je softwarová platforma, která má vlastní virtuální stroj, a je tak nezávislá na operačním systému nebo hardwaru počítače. Lze programovat v několika jazycích jako jsou Visual Basic, C++, J#, C# a další. Všechny tyto jazyky jsou při kompilaci překládány do Common Intermediate Language (CIL), který se pak dále kompiluje přímo do strojového kódu procesoru. Jazyk C# byl vytvořen speciálně pro tuto platformu. Je to objektově orientovaný jazyk, jehož syntaxe vznikla na základě jazyků C++, Java nebo Delphi. Více informací o tom jazyce lze nalézt v [5].

Architektura .NET je zobrazena na obrázku 1. Nad operačním systémem stojí Common Language Runtime (CLR), který je virtuálním strojem pro tuto platformu. Obsahuje Just-In-Time Compiler, který překládá spouštěný program z CIL do nativního kódu operačního systému. Další součástí je Base Class Library, která obsahuje základní třídy potřebné pro vývoj aplikací. Součástí jsou i třídy pro práci s daty a XML či vytváření webových a Windows aplikací. Je dostupná pro všechny jazyky platformy .NET. Projekty v .NET mohou být různých typů např. knihovna tříd, Windows aplikace, konzolová aplikace a jiné. Tyto projekty se pak kompilují do tzv. assembly, kterými mohou být soubory s příponou .exe nebo .dll, kde záleží zda jsou spustitelné, nebo slouží jako knihovna pro jiné assembly.



Obrázek 1: Architektura .NET

Spolu s instalací CLR na konkrétní počítač se instaluje také Global Assembly Cache (GAC), která má v sobě uloženy assembly, které jsou sdíleny více aplikacemi tohoto počítače. Implicitně jsou zde uloženy assembly obsahující třídy Base Class Library. Do GAC je možné přidávat i vlastní třídy, ovšem je nutné mít práva administrátora a assembly, která má být přidána, musí obsahovat silný název, který se skládá z textového názvu, čísla verze, veřejného klíče a digitálního podpisu. Assembly obsažena v GAC mají uděleny plnou důvěru a mohou přistupovat k jakýmkoliv zdrojům. Jejich důvěryhodnost je ověřována pouze při nahrávaní.

Současná verze této platformy je 3,5. Oproti předchozím verzím byla přidána např. lepší práce s daty.

3 Bezpečnost v .NET

Bezpečnost je v dnešní době stále častěji se opakující slovo, což je způsobeno potřebou uchovat data ale také běh aplikace v neporušené formě. Proto se i tato práce otázkou bezpečnosti bude zabývat. Jelikož cílem práce je vyvinout systém pro opravu projektů a při této opravě bude nutné spouštět řešení úloh studenty, je nutné zabezpečit tento systém tak, aby nedošlo ke zhroucení systému nebo dokonce serveru, na kterém tento systém poběží. Proto se práce bude zabývat hlavně bezpečností spouštění cizího kódu, u kterého nelze předvídat jeho chování.

Platforma .NET má vytvořený poměrně dobrý způsob zabezpečení. Hlavní klíčové koncepty bezpečnosti .NET jsou tyto:

- Zabezpečení přístupu ke kódu (CAS) - zabývá se zdrojovým kódem a oprávněními, které jsou mu povoleny popř. zakázány. Toto zabezpečení zahrnuje kódové skupiny, úrovně zabezpečení a sady oprávnění, čímž se budeme zabývat v následujících kapitolách.
- Zabezpečení založené na evidenci - využívá se k zjišťování informací o kódu a hostujícího prostředí aplikační domény a je nezbytné pro další části systému zabezpečení jako je např. zabezpečení přístupu ke kódu.
- Zabezpečení založené na rolích - věnuje se uživatelům a jejich oprávněním.
- Deklarativní a imperativní zabezpečení - dva způsoby, jak kódu nebo uživatelům přidělit nebo odebrat nějaké oprávnění.
- Kryptografie - důležitý článek zabezpečení, využívá se např. při digitálním podpisu assembly, ale také k dalším úkonům. Touto částí zabezpečení se tato práce nebude zabývat.

Všechny tyto koncepty spolu spolupracují a tvoří dohromady propracovaný systém zabezpečení. Tento bezpečnostní systém je podřízen operačnímu systému, kde jsou oprávnění obvykle založená na uživateli nebo roli. Takže se může stát, že i přesto že .NET udělil kódu požadované oprávnění, tak toto oprávnění může být zamítnuto operačním systémem a nemusí tak tento program běžet bez chyby.

Jmenný prostor *System.Security* poskytuje základní strukturu pro bezpečnostní systém .NET. Zahrnuje rozhraní, atributy, výjimky a bázové třídy pro oprávnění. Součástí tohoto jmenného prostoru je také třída *CodeAccessPermission*, která definuje základní strukturu všech tříd pro oprávnění přístupu ke kódu. Důležitou výjimkou v tomto jmenném prostoru je bezpečnostní výjimka *SecurityException*, která je vyhozena pokaždé, když se kód pokouší provést nějakou operaci nebo přistoupit ke zdrojům a nemá na to potřebná oprávnění.

Bezpečnostní model se skládá z několika částí, kterými jsou úrovně zabezpečení, kódové skupiny, sady oprávnění, evidence assembly a evidence hostující aplikační domény, ve které je kód spouštěn.

3.1 Evidence

Evidence je soubor informací o kódu nebo assembly zjištěné bezpečnostním systémem. Tyto informace jsou poskytovány zavaděčem (Loader) nebo důvěryhodným zdrojem, kterým je CLR nebo hostující prostředí aplikační domény.

Aplikační doména představuje oddělený běh .NET aplikace uvnitř CLR procesu. Izolace domén je založená na vlastnosti bezpečného oddělení paměti, protože aplikace z jiné domény nemůže přímo přistupovat k cizímu adresovému prostoru. Hostitelem aplikační domény mohou být:

- Browser host (Internet Explorer) - spouští částečně důvěryhodný kód v kontextu webového prohlížeče.
- Server host (ASP.NET) - spouští kód webové aplikace.
- Shell host - spouští řízené aplikace (.exe soubory) z příkazového řádku Windows.
- Custom hosts - Aplikace, která se spouští v CLR.

Evidence se dělí na dva typy, kterými jsou důvěryhodná a nedůvěryhodná evidence. Důvěryhodná evidence se skládá z informací zjištěných správcem zásad zabezpečení nebo důvěryhodným hostitelem, který inicializoval běhové prostředí („host-provided evidence“), tímto typem evidence může být např. digitální podpis. Nedůvěryhodná evidence se skládá z informací uložených přímo v assembly („assembly provided evidence“) autorem kódu. Informace uložené v tomto typu evidence musí být nejdříve ověřeny, než budou použity, nebo jim bude důvěřováno.

Když CLR a hostitel zjistí všechny informace o assembly, tak je zabalí do jediné kolekce typu *Evidence*. Evidenci pak CLR používá k rozhodnutí o zabezpečení a udělování oprávnění. Evidence může obsahovat několik částí:

- Aplikační adresář - adresář, ve kterém je aplikace nainstalována
- Hash - kryptografický hash, např. SHA1
- Vydavatel - digitální podpis vydavatele kódu
- Server - umístění odkud assembly pochází, např. http://www.microsoft.com
- Silný název - kryptografický silný název assembly
- URL - URL zdrojového adresáře, odkud assembly pochází, pokud je stažena z internetu
- Zóna - zóna, odkud assembly pochází, např. internet

Evidence je součástí každé assembly, indikuje běhovému prostředí, že kód má jednotlivé charakteristiky. O evidenci se můžete dočíst také v [7], [6] a [15].

Na základě vlastností uložených v evidenci se assembly zařazují do kódových skupin, které mají hierarchickou strukturu a jsou uloženy na několika úrovních.

3.2 Úrovně zabezpečení

Úrovně a jejich umístění v počítači, jak je uvádí [7] jsou:

- Enterprise - %Systemroot%\Microsoft.NET\Framework\version\Config\enterprise.config
- Machine - %Systemroot%\Microsoft.NET\Framework\version\Config\security.config
- User - %UserProfile%\Application Data\Microsoft\CLR Security Config\version\security.config
- Domain - N/A

Úroveň *Enterprise* odpovídá podnikové nebo rozlehlé síti. Úroveň *Machine* zahrnuje počítač, ve kterém je kód spouštěn. Nejčastěji se konfiguruje právě tato úroveň spolu s další úrovní *User*, která odpovídá uživatelskému profilu uživatele, který aplikaci spouští. Úroveň *Domain* odpovídá aplikační doméně spouštěné aplikace a existuje pouze za běhu této aplikace, musí být specifikována programově a neexistuje pro ni žádný konfigurační soubor.

Pro každou z těchto úrovní jsou vytvořeny kódové skupiny a k nim přiřazeny sady oprávnění. Výsledná oprávnění přiřazená assembly jsou pak průnikem oprávnění přiřazených na jednotlivých úrovních. Tento způsob nastavení oprávnění je velmi výhodný, protože je možné např. odebrat práva nějakému uživateli tak, že změníme konfiguraci na uživatelské úrovni, tzn. překryjeme původní nastavení z vyšších úrovní.

3.3 Sady oprávnění

Sady oprávnění určují, co bude moci kód dané assembly provádět. Na každé úrovni jsou implicitně vytvořeny tyto sady oprávnění:

- FullTrust - sada oprávnění, která uděluje kódu neomezený přístup ke všem chráněným prostředkům. Tuto sadu by měly mít pouze plně důvěryhodné assembly.
- SkipVerification - zahrnuje jediné oprávnění, které přiděluje právo na přeskočení ověřování. Assembly s tímto oprávněním není prověřována bezpečnostním systémem.
- Execution - zahrnuje jediné oprávnění, které umožňuje provádění aplikace.
- Nothing - odepírá všechna oprávnění a to včetně oprávnění *Execution* pro provádění aplikace.
- LocalIntranet - obsahuje výchozí oprávnění přidělena všem aplikacím místního intranetu.
- Internet - obsahuje výchozí oprávnění přidělena všem internetovým aplikacím.
- Everything - obsahuje všechna oprávnění s výjimkou oprávnění *SkipVerification* pro přeskočení ověřování.

Sady oprávnění je možno změnit, smazat, nebo také vytvořit nové.

3.4 Oprávnění

Sady oprávnění se skládají z jednotlivých typů oprávnění, při čemž každá sada oprávnění obsahuje alespoň jeden typ oprávnění. Existují tři druhy oprávnění a každý z nich má specifický účel:

- Oprávnění přístupu ke kódu - reprezentuje přístup k chráněným prostředkům nebo schopnost provádět chráněné operace. Dále se jím budeme zabývat v 3.4.1.
- Oprávnění založené na identitě - vyjadřuje, že kód je pověřen podporovat některé druhy identit. Více se dozvíme v 3.4.2.
- Oprávnění založené na rolích - poskytuje mechanismus autentizace, kdy chování aplikace záleží na tom, v jaké specifické roli se uživatel nachází, popř. jaká je jeho identita. Další informace naleznete v 3.4.3.

3.4.1 Oprávnění přístupu ke kódu

Zajišťuje řízení přístupu k chráněným prostředkům, kterými může být síť, místní disk, proměnné prostředí atd., nebo k provádění chráněných operací, kterými může být např. přístup k neřízenému kódu. Cílem je, aby spuštěný kód přistupoval pouze ke zdrojům, ke kterým má udělena oprávnění přistoupit, a prováděl pouze operace, které mu jsou povoleny. Všechna oprávnění pro přístup k chráněným zdrojům jsou odvozena od třídy *CodeAccessPermission*, která poskytuje metody pro vyžádání, povolení a zakázání oprávnění, a dále také pro sjednocení, rozdíl a zjištění, zda je oprávnění podmnožinou jiného. Těmito metodami se budeme zabývat v kapitole 3.9.

.NET Framework poskytuje tyto oprávnění pro přístup k chráněným zdrojům:

- AspNetHostingPermission - oprávnění přístupu ke zdrojům v prostředí ASP.NET
- DirectoryServicesPermission - oprávnění pro přístup ke třídám ve jmenném prostoru *System.DirectoryServices*, který obsahuje adresářové služby jako např. Active Directory
- DnsPermission - oprávnění pro práci se službou DNS
- EnvironmentPermission - oprávnění pro práci s proměnnými prostředí
- EventLogPermission - oprávnění pro práci z protokolem událostí
- FileDialogPermission - oprávnění přístupu k souborům, které byly uživatelem vybrány v dialogu pro otevření souborů
- FileIOPermission - oprávnění pro práci se soubory a adresáři
- IsolatedStorageFilePermission - oprávnění pro přístup k izolovanému úložišti, které je asociované se specifickým uživatelem a s některým aspektem identity kódu jako je webová stránka, vydavatel nebo digitální podpis

- MessageQueuePermission - oprávnění pro práci s frontou zpráv přes rozhraní Microsoft Message Queuing
- OdbcPermission - oprávnění pro přístup k datovému zdroji přes Open Database Connectivity
- OleDbPermission - oprávnění přístupu k databázím pomocí rozhraní OLE DB (Object Linking and Embedding Database)
- OraclePermission - oprávnění pro přístup k databázi Oracle
- PerformanceCounterPermission - oprávnění pro práci s čítačem výkonu
- PrintingPermission - oprávnění k tisku
- ReflectionPermission - oprávnění k používání mechanismu reflexe, který poskytuje informace o typech za běhu programu
- RegistryPermission - oprávnění přístupu k registrům systému Windows
- SecurityPermission - oprávnění ke spouštění kódu, využívání mechanismu zabezpečení, volání neřízeného kódu, přeskočení ověřování aj.
- ServiceControllerPermission - oprávnění pro spouštění a zastavování služeb
- SocketPermission - oprávnění k přijímání či vytváření spojení transportní vrstvy protokolu TCP/IP
- SqlClientPermission - oprávnění k přístupu k SQL databázím
- UIPermission - oprávnění k využívání uživatelského rozhraní
- WebPermission - oprávnění k přijímání či vytváření spojení vrstvy protokolu HTTP

Pokud by tato oprávnění nebyla dostačující, nabízí .NET abstraktní třídy, ze kterých je možné vytvořit vlastní oprávnění. Těmito abstraktními třídami jsou *DBDataPermission* pro přístup k databázím, *IsolatedStoragePermission* pro přístup k izolovaným úložištím a *ResourcePermissionBase* pro přístup k systémovým zdrojům.

Další informace k jednotlivým oprávněním se lze dozvědět v [6].

3.4.2 Oprávnění založené na identitě

Oprávnění na základě identity pomáhají chránit od neautorizovaného přístupu. CLR uděluje oprávnění založené na identitě na základě informací známých o assembly (evidenci - viz 3.1). Každé oprávnění na základě identity reprezentuje samostatný druh evidence, který musí mít assembly, aby mohla správně fungovat. Tyto oprávnění jsou pak přidělena podle toho, zda kód obsahuje příslušnou evidenci.

Protože oprávnění na základě identity mají část funkcionality společnou s oprávněními přístupu ke kódu, jsou tyto oprávnění odvozeny od stejné bázové třídy jako oprávnění přístupu ke kódu a to *CodeAccessPermission*.

.NET Framework poskytuje tyto oprávnění na základě identity:

- PublisherIdentityPermission - digitální podpis
- SiteIdentityPermission - internetové stránky, odkud assembly pochází
- StrongNameIdentityPermission - silný název assembly
- URLIdentityPermission - URL odkud assembly pochází (zahrnuje prefix protokolu - http, https, ftp, atd.)
- ZoneIdentityPermission - zóna, odkud assembly pochází

Podrobnější informace o těchto oprávněních se nachází v [6].

3.4.3 Oprávnění založené na rolích

Oprávnění založené na rolích zahrnuje pouze jediné oprávnění, kterým je *PrincipalPermission*. Toto oprávnění se zabývá ověřováním, zda aktivní uživatel je ve specifické roli nebo má specifickou identitu, která je představována typem implementující rozhraní IPrincipal. Na rozdíl od předchozích druhů oprávnění *PrincipalPermission* implementuje pouze rozhraní IPermission, které poskytuje pouze metody pro vyžádání, sjednocení, rozdíl oprávnění apod. Další informace lze nalézt v [6].

3.5 Kódové skupiny

Sady oprávnění jsou přiřazeny kódovým skupinám, při čemž jedna sada oprávnění může být přiřazena více kódovým skupinám. Do těchto kódových skupin pak jsou přiřazovány jednotlivé assembly na základě podmínek členství. Toto členství se určuje s pomocí vlastností assembly, kterými jsou již zmíněné zóna, silné jméno assembly, vydavatel apod., které se nachází v evidenci. Avšak může být vytvořena i vlastní podmínka členství, která se definuje XML souborem. Assembly může být zařazena i do více kódových skupin, pokud splňuje jejich podmínky členství.

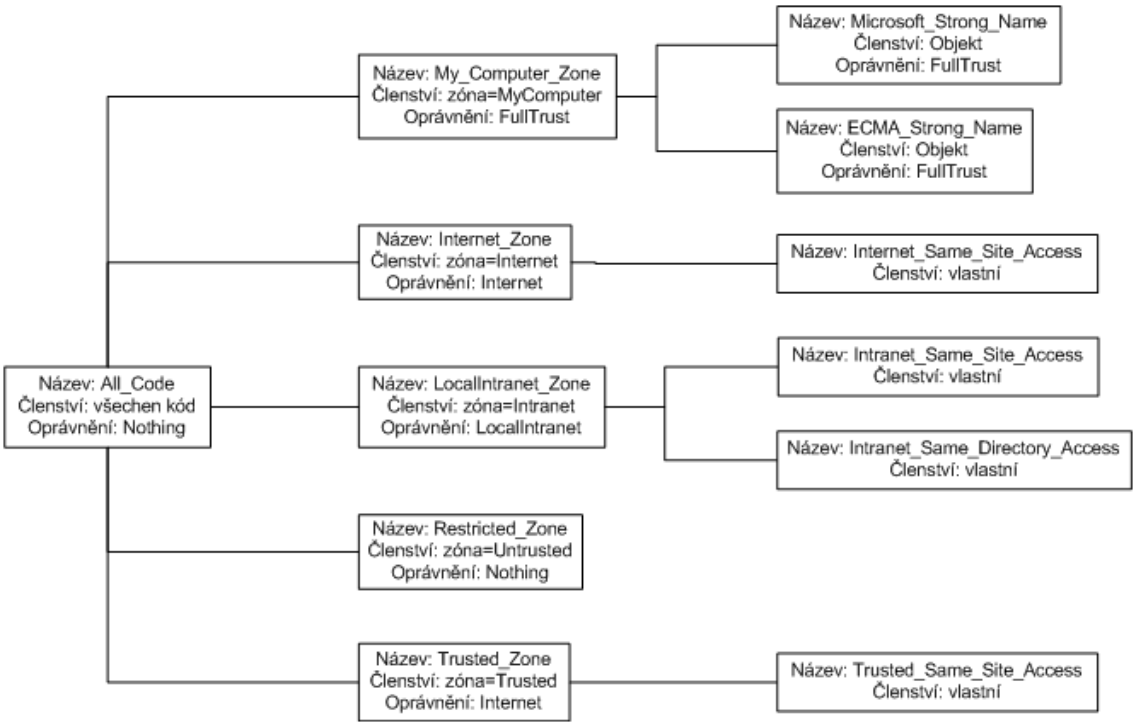
Na úrovni *Machine* je implicitně vytvořeno několik kódových skupin. Tyto kódové skupiny jsou založeny na zónách internetu tak, jak je obsahuje Internet Explorer. Těmito zónami jsou:

- My computer (Počítač) - tato zóna obsahuje aplikace pocházející z počítače, ve kterém jsou spouštěny.
- Internet - obsahuje všechny weby v internetu s výjimkou těch, které jsou uvedeny jako důvěryhodné servery nebo servery s omezeným přístupem.
- LocalIntranet (Místní síť) - obsahuje weby, které se nacházejí v místní síti.

- Trusted (Důvěryhodné servery) - seznam webů, kterým byla udělena důvěra, že nepoškodí počítač ani soubory.
- Untrusted (Servery s omezeným přístupem) - seznam webů, které mohou poškodit počítač.

Podmínkou členství kódových skupin na úrovni Machine jsou tedy výše uvedené zóny. Hlavní kódovou skupinou je *All_Code*, do které je přiřazen veškerý kód, tato skupina tvoří kořen stromové struktury kódových skupin a přiřazuje sadu oprávnění *Nothing*, což znamená, že nepřiřazuje žádná práva. Skupina *All_Code* obsahuje tyto podskupiny:

- My_Computer_Zone - podmínkou členství je zóna *My computer*, jsou do ní přiřazeny všechny aplikace umístěné v počítači. Této kódové skupině je přiřazena sada oprávnění *FullTrust*.
  - Microsoft_Strong_Name - podmínka členství je objekt podepsaný silným názvem firmy Microsoft (Microsoft Strong Name). Přiřazená sada oprávnění skupiny je rovněž *FullTrust*.
  - ECMA_Strong_Name - podmínka členství je objekt podepsaný silným názvem ECMA (ECMA Strong Name). Sada oprávnění přidělená této skupině je taktéž *FullTrust*.
- LocalIntranet_Zone - podmínkou členství je zóna *Intranet*, tudíž jsou do ní přiřazeny aplikace místního intranetu. Přidělenou sadou oprávnění je *LocalIntranet*.
  - Intranet_Same_Site_Access - obsahuje oprávnění pro připojení se zpět na intranet, odkud daný kód pochází.
  - Intranet_Same_Directory_Access - obsahuje oprávnění pro čtení z adresáře, odkud kód pochází.
- Internet_Zone - podmínkou členství je zóna *Internet*, jsou do ní přiřazeny aplikace pocházející z internetu. Sada oprávnění pro tuto kódovou skupinu je *Internet*.
  - Internet_Same_Site_Access - obsahuje oprávnění pro připojení se zpět na internet, odkud daný kód pochází.
- Restricted_Zone - podmínkou členství je zóna *Untrusted*, jedná se o nedůvěryhodný kód, kterému je přidělena sada oprávnění *Nothing*.
- Trusted_Zone - podmínkou členství je zóna *Trusted*. Tato kódová skupina obsahuje sadu oprávnění *Internet*.
  - Trusted_Same_Site_Access - obsahuje oprávnění pro připojení se zpět na web, odkud daný kód pochází.



Obrázek 2: Struktura kódových skupin na úrovni *Machine*

Jejich hierarchická struktura pak lze vidět na obrázku 2.

Na úrovních *User* a *Enterprise* je implicitně vytvořena pouze skupina *All_Code*, které je přiřazena sada oprávnění *FullTrust*.

Kódové skupiny mohou být označeny atributy *Exclusive* a *LevelFinal*. Atribut *Exclusive* určuje, že assembly nedostane více oprávnění než je přiřazeno ke kódové skupině, která má tento atribut. Assembly nemůže být zařazena do více kódových skupin majících atribut *Exclusive* na stejné úrovni. Atribut *LevelFinal* určuje, že žádná z nižších úrovní kromě Domain nebude brána v úvahu při zařazování do kódových skupin a vyhodnocování oprávnění.

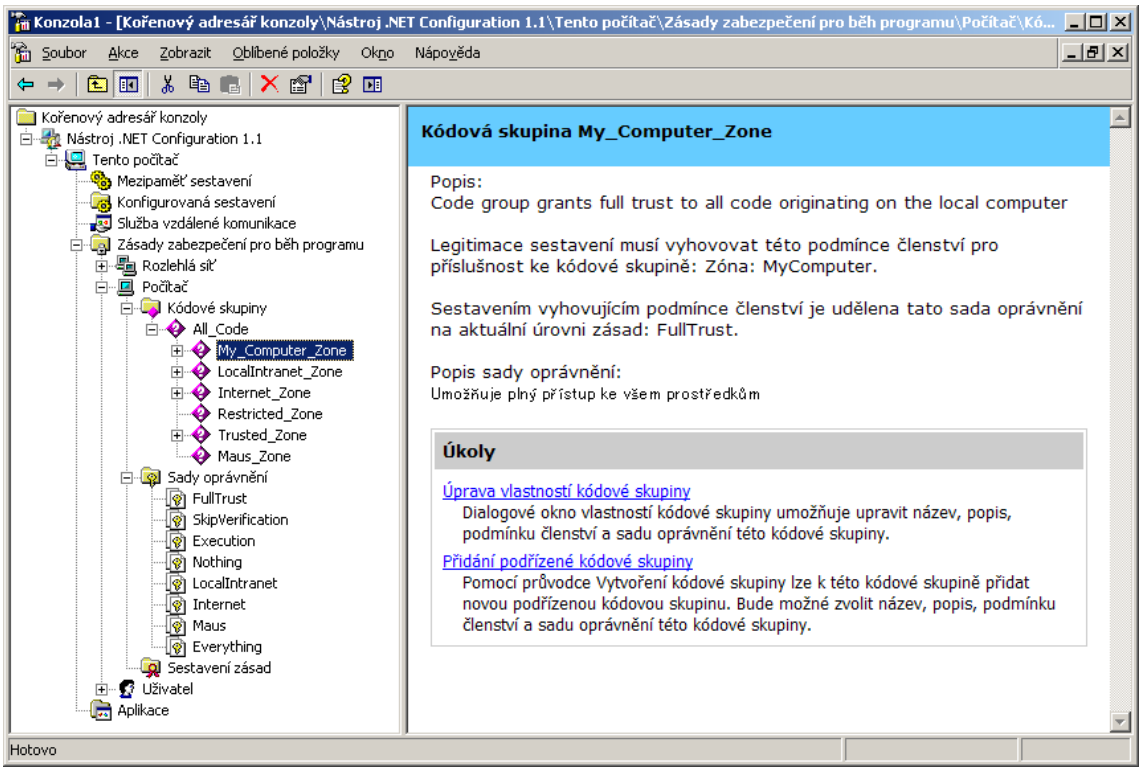
Výsledná oprávnění konkrétní úrovně jsou určena sjednocením oprávnění přiřazených kódovým skupinám, jejichž je assembly členem.

3.6 Správa kódových skupin a sad oprávnění

Kódové skupiny a sady oprávnění se dají spravovat několika způsoby, a to pomocí modulu *snap-in konzole MMC*, pomocí utility pro příkazový řádek *caspol.exe*, nebo přímo v XML souborech, ve kterých je toto nastavení uloženo. Oba nástroje (caspol.exe a snap-in konzole MMC) umožňují nejen zobrazení, přidání, změnu nebo smazání zásad zabezpečení, kódových skupin a sad oprávnění, ale také analýzu bezpečnostních nastavení konkrétní assembly. Při této analýze se lze dozvědět, do kterých kódových skupin je as-

sembly přiřazena a jaké má tedy přiděleny oprávnění. Více k tomuto téma se lze dozvědět v [20].

Použití modulu snap-in konzole MMC, kterou můžete vidět na obrázku 3, má nespornou výhodou v grafickém uživatelském rozhraní, které umožňuje lepší přehled a pohodlnější práci. Na druhou stranu, pokud byste potřebovali provést stejné nastavení na více počítačích je lepší provést toto nastavení použitím utility caspol.exe (Code Access Security Policy), kde lze využít výhody příkazového řádku a např. vytvořit skript pro tyto nastavení. Použití utility caspol.exe lze vidět na obrázku 4.

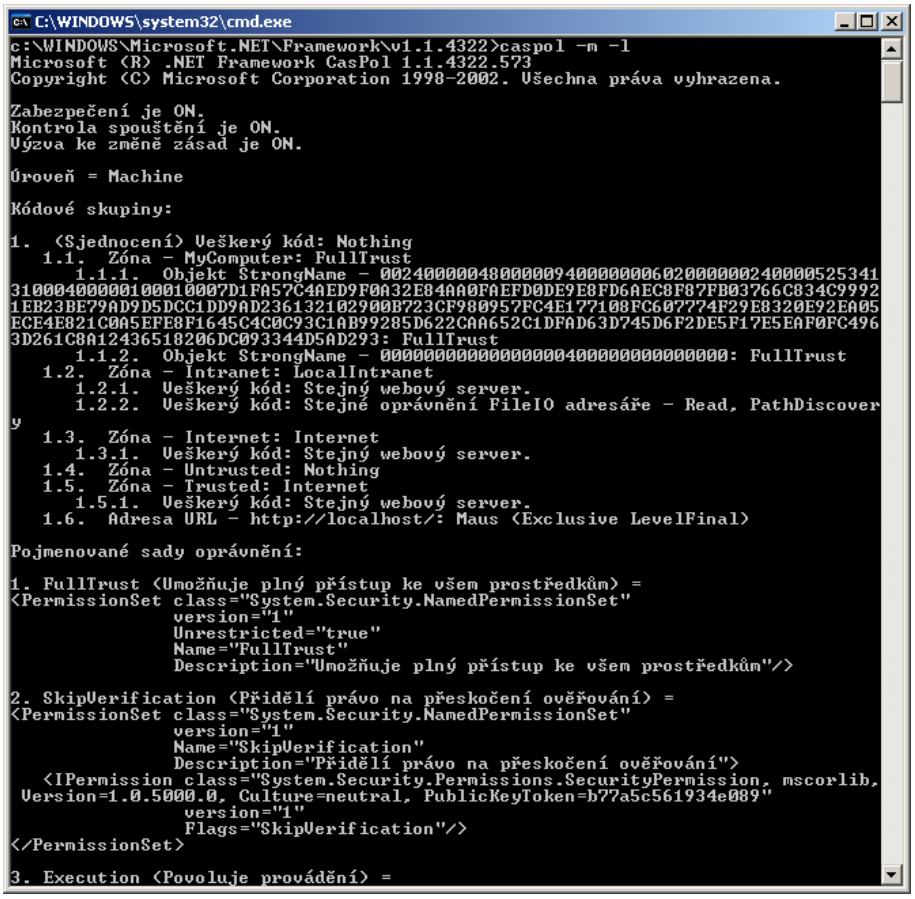


Obrázek 3: Modul snap-in konzole MMC

3.7 Princip načítání assembly

Při spouštění řízeného kódu se děje spousta kroků, o kterých uživatel vůbec neví, pokud nedojde k nějakému selhání a není mu vyhozena nějaká výjimka. Těmito kroky jsou nahrávání assembly, vyhodnocení bezpečnostních zásad bezpečnostním systémem, Just-in-time kompilace, ověření typové bezpečnosti a nakonec povolení spuštění. Tento proces je znázorněn na obrázku 5, který byl převzat z [13].

Každá řízená aplikace běží v běhovém prostředí s kontextem hostitele (aplikační domény). Hostitel je důvěryhodná část kódu, která je zodpovědná za spuštění běhového prostředí, specifikaci podmínek, za kterých se běhové prostředí spouští, a kontrolu vztahů

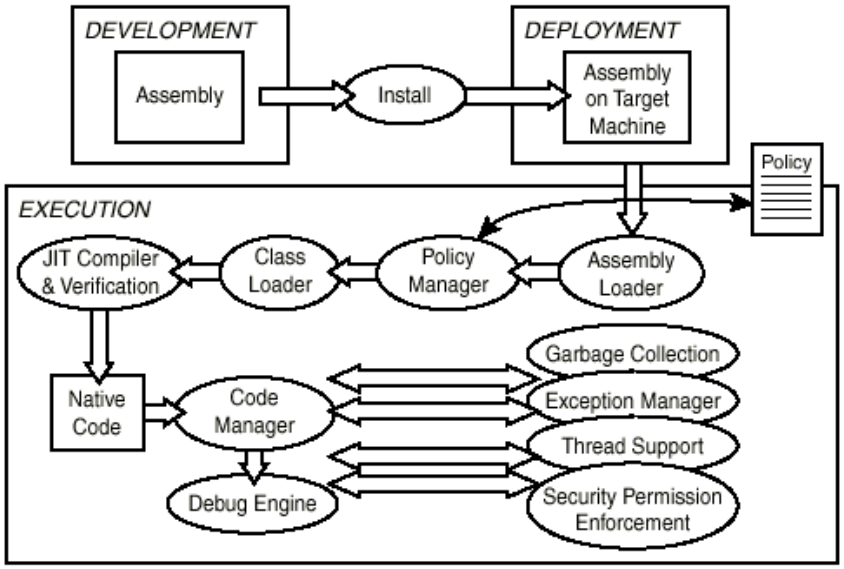


Obrázek 4: Utilita caspol.exe

ve spouštěném kódu. Po spuštění běhového prostředí je načten vstupní bod aplikace (u desktopových aplikací metoda *Main*) a předává se kontrola tomuto vstupnímu bodu, aby mohlo dojít ke spuštění aplikace.

Prvním krokem načítání assembly je její lokalizace. Obvykle bývají assembly umístěné na disku, nebo stažené ze sítě, ale je také možné, že jsou načítány dynamicky z pole bytů. Po té, co je assembly lokalizována, je předána zavaděči assembly (Runtime's Assembly Loader), který prochází její obsah a vytváří datové struktury, které reprezentují obsah assembly běhovému prostředí. Řízení je po té předáno správci zásad bezpečnostního systému.

Zavaděč assembly je také zodpovědný za analýzu referencí na soubory jednotlivých assembly. Assembly se vždy skládá alespoň z jednoho souboru, ale tento soubor může obsahovat reference na další podřízené soubory, které dohromady tvoří assembly. Tyto reference jsou obsaženy v tzv. manifestu assembly, který je uložen v prvním souboru assembly. Manifest obsahuje pro každou referenci kryptografický hash obsahu souboru, na který odkazuje. Při analýze reference je nalezen tento soubor, spočítán jeho hash



Obrázek 5: Princip načítání assembly z [13]

a porovnán s hodnotou uloženou v manifestu. Pokud toto porovnání uspěje, je tento soubor načten, v opačném případě načítání selže.

Správce zásad bezpečnostního systému je jádrem zabezpečení běhového prostředí. Jeho prací je rozhodnout, která oprávnění budou přidělena jednotlivým assembly při načítání běhovým prostředím. Před tím než je kód assembly spuštěn, rozhodne správce zásad zabezpečení, zda bude kód vůbec spuštěn a pokud ano, tak jaká práva bude mít.

Vstupy pro správce zabezpečení jsou aktuální zásady zabezpečení na jednotlivých úrovních, evidence assembly a množina požadovaných oprávnění na úrovni assembly.

CLR uděluje oprávnění aplikační doméně a assembly. Proces přidělování práv zahrnuje jeden nebo oba následující kroky:

- Výpočet povolených sad oprávnění - při načítání běhové prostředí zjišťuje sady oprávnění přidělených na všech úrovních. Běhové prostředí udělá průnik opatření přes všechny úrovně, přičemž vznikne jedna výsledná sada oprávnění assembly nebo aplikační domény.

- Výpočet udělených oprávnění - běhové prostředí porovná konečnou sadu povolených oprávnění s oprávněními, které assembly požaduje, výsledkem jsou oprávnění udělená assembly. Tento krok neplatí pro aplikační domény.

Když jsou splněny minimální požadavky assembly a assembly má udělena oprávnění pro spuštění je řízení předáno zavaděči tříd (Class Loader). Třídy assembly se nahrávají líně, tzn. nahrávají se jen ty třídy, které jsou potřeba. Zavaděč tříd je zodpovědný za načtení tabulek metod a datových struktur asociovaných s třídou do paměti a ověření viditelnosti tříd a rozhraní. Po inicializaci datových struktur tříd je připraven přístup k individuálním metodám třídy.

Po načtení třídy zavaděčem tříd dochází k ověření typové bezpečnosti MSIL uvnitř assembly a generování nativního kódu. To je práce pro Just-In-Time Compiler. Metody tříd jsou také načítány, ověřovány a překládány líným způsobem. Když je metoda v procesu poprvé zavolána, je zkontrolována typová bezpečnost metody a pak je převedena do nativního kódu.

JIT Compiler také hraje roli v ohodnocení deklarativní bezpečnosti na úrovni tříd a metod, kterými může být samotné vyžádání nějakého oprávnění, nebo také vyžádání oprávnění pro dědice a volající kód (viz kapitola 3.9.12).

Po té, co byl úspěšně vygenerován nativní kód, je připraveno jeho spuštění. Při spuštění se mohou objevit reference na nezpracované třídy, metody a assembly. Tyto reference způsobí opětovné volání JIT Compiler, zavaděče tříd a zavaděče assembly, a pokud to bude nutné tak i správce zásad zabezpečení.

Další informace ohledně nejen načítání ale i celého životního cyklu assembly se můžete dozvědět v [13].

3.8 Procházení zásobníku (StackWalk)

Procházení zásobníku je operace, kterou provádí správce zabezpečení, když zjišťuje, zda má kód potřebná oprávnění. K této operaci dochází pokaždé, když se vyskytne nějaký požadavek na oprávnění imperativním či deklarativním způsobem, nebo implicitně voláním nějaké třídy .NET Framework, která je automaticky chráněna bezpečnostním systémem. Běhové prostředí ověřuje, že každý v řetězci volání (každá předchozí metoda, která volala danou metodu) má potřebná oprávnění k provedení požadované operace. Toto chrání před útokem, kdy privilegovaná část kódu je zneužita nebezpečným kódem, který nemá potřebná oprávnění, protože před samotným spuštěním kódu je ověřen řetězec volání, kde se zjistí, že původce volání, kterým je tento nebezpečný kód, nemá příslušná oprávnění a je vyhozena bezpečnostní výjimka.

Toto chování se však dá obejít programovým použitím CAS, kterým se bude zabývat kapitola 3.9, kdy můžeme kódu nějaká oprávnění přidělit, nebo odebrat, a tudíž pak nebude docházet k procházení celého řetězce, ale procházení se zastaví u tohoto přidání nebo odebrání, na jejichž základě bude požadovaná operace provedena, nebo bude vyhozena bezpečnostní výjimka. Povolení nebo odebrání nějakého oprávnění může být v rámci metody provedeno pouze jednou. Pokud chceme provést jiné povolení nebo odebrání musíme předchozí nastavení vrátit, a pak je teprv možné provést nové nastavení. Na konci metody je doporučeno všechny změny vrátit. K tomuto tématu se lze dočíst také v [17] a [6].

3.9 Programové použití CAS

Kromě výše uvedeného způsobu nastavení bezpečnosti pomocí kódových skupin a sad oprávnění je možné nastavit oprávnění ještě programově přímo v kódu. Lze to udělat dvěma způsoby a to deklarativně nebo imperativně. Hodně věcí lze nastavit oběma způsoby, ovšem každý z těchto způsobů umožňuje i některá nastavení, která nejsou tím druhým způsobem uskutečnitelná. Imperativní způsob se provádí využitím tříd systému

CAS přímo v kódu a jeho výhodou je, že umožňuje nastavení práv za nějakých podmínek, které jsou známy až za běhu aplikace, a zároveň umožňuje reagovat na situace, kdy kód nemá dostatečná oprávnění. Naproti tomu deklarativní způsob se provádí pomocí atributů, je tak o něco jednodušší, nastavuje oprávnění již v době kompilace, ukládá se v metadatech a je možné je získat pomocí mechanismu reflexe. Také je možné tyto oprávnění zjistit pomocí nástrojů jako je permview.exe. Další informace o programovém použití CAS se lze dočíst také v [10], [11], [6], [14] a [16].

Třídy systému CAS se nacházejí v jmenném prostoru *System.Security*, tyto třídy zpravidla implementují rozhraní *IPermission* nebo *IStackWalk*. Obě tyto rozhraní obsahují metody pro vyžádání, povolení a zakázání oprávnění.

3.9.1 Vyžádání oprávnění imperativním způsobem

Vyžádání oprávnění imperativním způsobem se provádí pomocí metody *Demand*, která se volá na konkrétním objektu oprávnění. Má-li kód dostatečné oprávnění, proběhne volání v pořádku. Nemá-li toto oprávnění, pak bude vyhozena výjimka typu *SecurityException*. Ukázka tohoto vyžádaní oprávnění je ve výpisu kódu 1.

```
public class DemandExample {
    public static void FileIODemand() {
        IPermission perm = new FileIOPermission(FileIOPermissionAccess.Read, "C:\\");
        try {
            //vyzadame opravneni – pokud kod opravneni nema
            //je vyhozena vyjimka SecurityException
            perm.Demand();
            Console.WriteLine("Kod ma opravneni pro cteni z C:\\");
        } catch (SecurityException) {
            Console.WriteLine("Kod nema opravneni pro cteni z C:\\");
        }
    }
}
```

Výpis 1: Vyžádání oprávnění imperativním způsobem

3.9.2 Vyžádání oprávnění deklarativním způsobem

Vyžádání oprávnění deklarativním způsobem se provádí použitím atributu konkrétního typu oprávnění a specifikací akce, kterou chceme provést pomocí hodnoty *Demand* výčtu *SecurityAction* a uvedení dalších informací pro vyžádání oprávnění pomocí pojmenovaných parametrů. Ukázka takového vyžádání oprávnění je ve výpisu kódu 2.

```
[FileIOPermission(SecurityAction.Demand, PathDiscovery = "C:\\")]
public static void FileIODemand() {
    Console.WriteLine("Kod ma opravneni pro cteni z C:\\");
    //cteni z disku C:\
}
```

Výpis 2: Vyžádání oprávnění deklarativním způsobem

3.9.3 Dočasné povolení oprávnění imperativním způsobem

Dočasné povolení oprávnění imperativním způsobem se provádí použitím metody *Assert* na instanci konkrétního oprávnění, které implementuje rozhraní *IStackWalk*. Lze takto nastavit oprávnění, které kód původně neměl. Po provedení kódu, kvůli kterého bylo oprávnění přiděleno, je nutné oprávnění opět odebrat, aby nedošlo k narušení bezpečnosti kódem, který nemá mít oprávnění přístupu k těmto prostředkům. Opětovné odebrání oprávnění se uskutečňuje statickou metodou *CodeAccessPermission.RevertAssert*. V ukázce výpisu kódu 3 se předpokládá, že kód původně oprávnění pro zápis do souboru neměl.

```
public class AssertExample {
        public static void FileIOAssert() {
            string name = @"C:\soubor.txt";
          FileIOPermission perm = new FileIOPermission(FileIOPermissionAccess.Write, name);
          //docasne povolime zapis do souboru
          perm.Assert();
          StreamWriter sw = null;
          try {
              sw = new StreamWriter(name);
              sw.WriteLine("Text zapsany do souboru.");
          } finally {
              if (sw != null) {
                  sw.Close();
              }
          }
          //zrusime docasne povoleni pro pristup k souboru
          CodeAccessPermission.RevertAssert();
      }
}
```

Výpis 3: Dočasné povolení oprávnění imperativním způsobem

3.9.4 Dočasné povolení oprávnění deklarativním způsobem

Dočasné povolení oprávnění deklarativním způsobem se provádí využitím atributu oprávnění *SecurityAction.Assert* a nastavení podmínek pomocí pojmenovaných parametrů. Ve výpisu kódu 4 také předpokládáme, že metoda, která bude tuto metodu volat, nemá oprávnění pro přístup k souboru.

```
[FileIOPermission(SecurityAction.Assert, Write = @"C:\soubor.txt")]
public static void FileIOAssert() {
    using (StreamWriter sw = new StreamWriter(@"C:\soubor.txt")){
        sw.WriteLine("Text zapsany do souboru.");
    }
}
```

Výpis 4: Dočasné povolení oprávnění deklarativním způsobem

3.9.5 Dočasné odepření oprávnění imperativním způsobem

Kromě dočasného povolení nějakého oprávnění lze také dočasně odepřít oprávnění. Takže kód, který byl původně oprávněn pro přístup k danému zdroji, nyní toto oprávnění mít nebude a při přístupu k danému zdroji bude vyhozena výjimka *SecurityException*. Toto odepření se provádí pomocí metody *Deny* konkrétního oprávnění, které má být odepřeno. Lze odepřít kterékoliv oprávnění, které implementuje rozhraní *IStackWalk*. Dočasné odepření je nutno po provedení části kódu, který má mít odepřeno oprávnění, opět zvrátit pomocí statické metody *RevertDeny* třídy *CodeAccessPermission*, která opět umožní přístup k chráněnému zdroji. Ukázka dočasného odepření je ve výpisu kódu 5, kde je nejprve zakázán přístup k souboru na disku a po neúspěšném pokusu o čtení je přístup k souboru opět vrácen a další pokus o čtení tak už bude úspěšný.

```
public class DenyExample {
    public static void FileIODeny() {
        string name = @"C:\soubor.txt";
        //vytvoreni instance tridy zabezpeceni pristupu k souboru
        FileIOPermission perm = new FileIOPermission(FileIOPermissionAccess.Read, name);
        //docasne odepreni pristupu k souboru
        perm.Deny();
        FileStream fs = null;
        try {
            //vyhodi vyjimku
            fs = File.Open(name, FileMode.Open);
        } catch (SecurityException se) {
            Console.WriteLine(se);
        } finally {
            if (fs != null) {
                fs.Close();
            }
        }
        //vraceni docasneho odepreni pristupu k souboru
        CodeAccessPermission.RevertDeny();
        try {
            //nevyhodi vyjimku
            fs = File.Open(name, FileMode.Open);
        } catch (SecurityException se) {
            Console.WriteLine(se);
        } finally {
            if (fs != null) {
                fs.Close();
            }
        }
    }
}
```

Výpis 5: Dočasné odepření oprávnění imperativním způsobem

3.9.6 Dočasné odepření oprávnění deklarativním způsobem

Dočasné odepření oprávnění deklarativním způsobem se provádí využitím atributu oprávnění *SecurityAction.Deny* a nastavení podmínek pomocí pojmenovaných parametrů. Ve výpisu kódu 6 je ukázáno, jak odepřít přístup k souboru deklarativním způsobem.

```
[FileIOPermission(SecurityAction.Deny, Read = @"C:\soubor.txt")]
public static void FileIODeny() {
    string name = @"C:\soubor.txt";
    FileStream fs = null;
    try {
        fs = File.Open(name, FileMode.Open);
    } catch (SecurityException) {
        Console.WriteLine("Kod nema opravneni pro pristup k " + name);
    } finally {
        if (fs != null) {
            fs.Close();
        }
    }
}
```

Výpis 6: Dočasné odepření oprávnění deklarativním způsobem

3.9.7 Využití metody PermitOnly imperativním způsobem

Využití metody *PermitOnly* imperativním způsobem se provádí v případě, že chceme odepřít všechen přístup k chráněným zdrojům až na jeden konkrétní, na kterém je tato metoda zavolána. Ve výpisu kódu 7, který je převzat z [11], je vidět použití této metody, kdy je povolen pouze přístup k systémové proměnné „OS“, přístup k ostatním vyhodí výjimku *SecurityException*, dokud není zavolána metoda *CodeAccessPermission.RevertPermitOnly*, která vrátí původní nastavení oprávnění.

```
/// <summary>
/// Priklad na pouziti metody PermitOnly
/// </summary>
public static class PermitOnlyExample
{
    public static void TestPermitOnly()
    {
        //vytvoreni objektu opravneni
        EnvironmentPermission envPerm = new EnvironmentPermission(
            EnvironmentPermissionAccess.AllAccess, "OS");
        //zamezeni pristupu ke vsem ostatnim zdrojum
        envPerm.PermitOnly();
        //je povolen pristup pouze ke promenne OS - cteni bude uspesne
        string os = Environment.GetEnvironmentVariable("OS");
        Console.WriteLine(os);
        try
        {
            //tento pokus o precteni skonci vyjimkou
            string temp = Environment.GetEnvironmentVariable("TEMP");
            Console.WriteLine(temp);
```

```
            }
            catch(SecurityException)
            {
                Console.WriteLine("Pristup k promenne prostredi byl odepren");
            }
            Stream stream = null;
            try
            {
                //zkouska pro pristup k souboru taktez selze
                stream = File.Open("C:/file.txt", FileMode.Open);
            }
            catch(SecurityException)
            {
                Console.WriteLine("Pristup k souboru byl odepren");
            }
            finally
            {
                if (stream != null)
                {
                    stream.Close();
                }
            }
            //zruseni omezeni pristupu
            CodeAccessPermission.RevertPermitOnly();
            try
            {
                //tento pokus o precteni jiz bude uspesny
                string temp = Environment.GetEnvironmentVariable("TEMP");
                Console.WriteLine(temp);
            }
            catch (SecurityException)
            {
                Console.WriteLine("Pristup byl odepren");
            }
        }
}
```

Výpis 7: Využití metody *PermitOnly* imperativním způsobem z [11]

3.9.8 Využití metody PermitOnly deklarativním způsobem

Využití metody *PermitOnly* deklarativním způsobem funguje úplně stejně jako imperativním způsobem. Používá se atribut *SecurityAction.PermitOnly* a nastavení pojmenovaných parametrů konkrétního oprávnění. Příklad takového použití je uveden ve výpisu 8.

```
[EnvironmentPermission(SecurityAction.PermitOnly,All="OS")]
public static void TestPermitOnly() {
    string os = Environment.GetEnvironmentVariable("OS");
    Console.WriteLine(os);
}
```

Výpis 8: Využití metody PermitOnly deklarativním způsobem

3.9.9 Metoda IsSubsetOf

Metoda *IsSubsetOf* se používá, pokud potřebujeme zjistit, zda je nějaké oprávnění podmnožinou jiného.

```
public class IsSubsetOfExample {
    public static void FileIOIsSubsetOf() {
        //vytvoreni instance tridy zabezpeceni pristupu k souborum v adresari TEMP
        FileIOPermission perm1 = new FileIOPermission(FileIOPermissionAccess.AllAccess,
            Environment.GetEnvironmentVariable("TEMP"));
        //vytvoreni instance tridy zabezpeceni pristupu k souborum na disku C:\
        FileIOPermission perm2 = new FileIOPermission(FileIOPermissionAccess.AllAccess, "C:\\"
            );
        if (perm1.IsSubsetOf(perm2)) {
            Console.WriteLine("Opravneni perm1 je podmnozinou perm2, tudiz adresar TEMP
                se nachazi na disku C:\\");
        } else {
            Console.WriteLine("Opravneni perm1 neni podmnozinou perm2, tudiz adresar TEMP
                se nachazi na jinem disku");
        }
    }
}
```

Výpis 9: Využití metody *IsSubsetOf*

3.9.10 Třída PermissionSet

Třída *PermissionSet* slouží k vytvoření sady oprávnění na úrovni zdrojového kódu. Představuje kolekci oprávnění, která může obsahovat různé druhy oprávnění, se kterými pak lze pracovat stejně, jakoby to bylo jedno oprávnění, tzn. podporuje metody *Demand*, *Assert*, *Deny* atd. viz výše.

```
public class PermissionSetExample {
    public static void Set() {
        //vytvoreni sady opravneni
        PermissionSet set = new PermissionSet(null);
        //pridani jednotlivych opravneni do sady
        set.AddPermission(new EnvironmentPermission(EnvironmentPermissionAccess.Read,"
            TEMP"));
        set.AddPermission(new FileIOPermission(FileIOPermissionAccess.AllAccess, "C:\\soubor.
            txt"));
        try {
            //vyzadani vsech opravneni v sade
            set.Demand();
        } catch (SecurityException) {
            Console.WriteLine("Nepodarilo se vyzadat nejakeho z opravneni v sade");
        }
    }
}
```

Výpis 10: Využití třídy *PermissionSet*

3.9.11 Vyžadování oprávnění na úrovni assembly

Vyžadování oprávnění na úrovni assembly je dalším způsobem jak určit oprávnění, které assembly potřebuje a které naopak potřebovat nebude už při jejím nahrávání běhovým prostředím. Toto vyžadování se provádí deklarativním způsobem na úrovni assembly a využívají se k tomu atributy *RequestMinimum* (minimální oprávnění), *RequestOptional* (nepovinná oprávnění) a *RequestRefuse* (odmítnutá oprávnění) výčtu *SecurityAction*. Atribut *RequestMinimum* určuje, které oprávnění jsou nezbytně nutné pro spuštění assembly a bez nichž nemůže assembly korektně běžet. Atribut *RequestRefuse* naopak určuje, které atributy assembly nepotřebuje ke svému běhu, umožňuje tak omezit sadu oprávnění, které budou assembly přiděleny, což zabraňuje zneužití assembly k účelům, pro které nebyla původně určena. Oprávnění uvedena v atributu *RequestRefuse* assembly přidělena nebudou. Atribut *RequestOptional* určuje oprávnění, které assembly může potřebovat, ale je schopna běžet i bez nich. Použití atributu *RequestOptional* také způsobuje, že oprávnění, které nejsou jeho součástí, nebo nejsou explicitně vyžádána pomocí atributu *RequestMinimum*, budou odmítnuta stejně jako kdyby na ně byl použit atribut *RequestRefuse*. Použití těchto atributů je znázorněno ve výpisu kódu 11. Vyžadování oprávnění na úrovni assembly se obvykle zapisuje do souboru AssemblyInfo.cs, který je vygenerován pro každou assembly vývojovým prostředím Visual Studio .NET.

```
// vyzaduje alespon umozneni cteni z adresare C:\Temp
[assembly:FileIOPermission(SecurityAction.RequestMinimum, Read = "C:\\Temp")]
// vyzaduje neomezeny pristup pro praci se soubory a adresari a odmita vsechna ostatni opravneni
[assembly:FileIOPermission(SecurityAction.RequestOptional, Unrestricted = true)]
// zamezi pristupu do adresare C:\Windows
[assembly:FileIOPermission(SecurityAction.Deny, Write = "C:\\Windows")]
```

Výpis 11: Použití atributů pro vyžádání oprávnění na úrovni assembly

3.9.12 Vyžadování oprávnění pro dědice (inheritors) a volající kód (linkers)

Vyžadování oprávnění pro dědice a volající kód je prováděno pomocí atributů *InheritanceDemand* a *LinkDemand* výčtu *SecurityAction*. Atribut *LinkDemand* určuje, že přímý předchůdce, který volá tuto třídu, musí mít uděleno specifikované oprávnění. Atribut *InheritanceDemand* určuje, že třída, která bude dědit z této třídy, nebo metoda, která bude překrývat tuto metodu, musí mít uděleno specifikované oprávnění. Pokud by takové oprávnění neměly, tak by byla vyhozena bezpečnostní výjimka. Kontrola těchto oprávnění se provádí už při kompilaci. Ukázka použití těchto atributů je ve výpisu 12.

```
// trida  dedici  z  MyClass musi mit take neomezeny pristup k souborum a adresarum
[class:FileIOPermission( SecurityAction.InheritanceDemand, Unrestricted = true )]
class MyClass  {.....  }
// trida ,  ktera  vola  MyClass musi mit take neomezeny pristup k souborum a adresarum
[class:FileIOPermission( SecurityAction.LinkDemand, Unrestricted = true )]
class MyClass  {.....}
```

Výpis 12: Použití atributů pro vyžádání oprávnění pro dědice a volající kód

3.9.13 Security-Transparent Code

Atributem *SecurityTransparent* se označuje celá assembly. Takto označenému kódu pak nemůžou být uděleny další privilegia. Kód označený *SecurityTransparent* způsobí, že jakékoliv požadavky na oprávnění přesměruje na kód, kterým byl zavolán. Pokud nedůvěryhodný kód zavolá metodu označenou tímto atributem a ta potřebuje vyšší oprávnění, pak vyžádání oprávnění přejde zpět k nedůvěryhodnému kódu a toto vyžádání selže. Kód s tímto atributem rovněž nemůže modifikovat procházení zásobníku, protože nemůže využít volání metody *Assert*. Pokud stejný kód bude zavolán z důvěryhodného kódu, vyžádání oprávnění uspěje. Ukázka způsobu označení kódu jako transparentního je ve výpisu kódu 13.

```
//oznacuje assembly jako transparentni vuci bezpecnosti
[assembly: SecurityTransparent]
```

Výpis 13: Označení kódu jako transparentní

Při použití transparentní bezpečnosti se kód dělí na dvě části, kterými jsou *security-transparent* a *security-critical*, kde *security-critical* se stará o vyhodnocování privilegií a *security-transparent* zahrnuje manipulaci s daty a logiku aplikace. Implicitně je všechen kód považován za *security-critical*.

3.10 Zabezpečení založené na rolích

Zabezpečení založené na rolích umožňuje měnit chování kódu na základě role, ve které se uživatel nachází, nebo přímo na identitě uživatele. Častěji se však používá rolí. Pro reprezentaci rolí se používají objekty implementující rozhraní *IPrincipal*, které představují kontext zabezpečení uživatele. CLR automaticky připojuje tyto objekty k jednotlivým vláknům aplikace, a proto každé vlákno běží s bezpečnostním kontextem určitého uživatele. Existuje několik typů těchto objektů a jsou využívány na základě toho, s kterým způsobem ověřování je asociována daná aplikační doména. Mezi způsoby ověřování patří mechanismus systému Windows, služba .NET passport a ASP.NET, ale lze vytvořit i vlastní způsob implementací rozhraní *IPrincipal*. Rozhraní *IPrincipal* poskytuje metodu *IsInRole* pomocí které lze ověřit, zda se uživatel, jehož kontext je reprezentován konkrétním objektem, nachází v požadované roli. Pro reprezentaci identity se používají objekty implementující rozhraní *IIdentity*. K zjištění identity se používá vlastnost *Identity* na objektu, který reprezentuje roli. Ukázka použití zabezpečení založeného na rolích se nachází ve výpisu kódu 14.

```
public class RoleExample {
    public static void Main(string[] args) {
        //nastaveni zpusobu autentikace, v tomto pripade se jedna o Windows
        AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal);
        //ziskani objektu principala
        WindowsPrincipal principal = (Thread.CurrentPrincipal as WindowsPrincipal);
        //ziska seznam roli zabudovanych ve windows
        WindowsBuiltInRole[] wbir = (WindowsBuiltInRole[])Enum.GetValues(typeof(
            WindowsBuiltInRole));
```

```
        //projde role a zjisti , zda se uzivatel nachazi v dane roli
        foreach (WindowsBuiltInRole roleName in wbir) {
            if ( principal .IsInRole((WindowsBuiltInRole)roleName)) {
                Console.WriteLine("Uzivatel se nachazi v roli {0}.", roleName);
            }
        }
         // ziska identitu
        WindowsIdentity identity = WindowsIdentity.GetCurrent();
          // vytvori objekt principala
        principal = new WindowsPrincipal(identity);
        //zkontroluje roli pomoci vyctu
        if ( principal .IsInRole(WindowsBuiltInRole.Administrator)) {
            Console.WriteLine("Uzivatel je administrator.");
        } else {
            Console.WriteLine("Uzivatel neni administrator.");
        }
          if ( principal . Identity .Name == "MILI\\pis103") {
            Console.WriteLine("Ahoj Mili.");
        }
        Console.ReadLine();
    }
}
```

Výpis 14: Použití zabezpečení založeného na rolích

Stejně jako u zabezpečení přístupu ke kódu lze použít i deklarativní způsob, který lze vidět ve výpisu kódu 15.

```
[PrincipalPermission(SecurityAction.Demand, Role = "Administrator")]
public static void AdministratorsOnly() {
    //kod teto metody se provede pouze pokud je uzivatel v roli administratora
}
```

Výpis 15: Použití zabezpečení založeného na rolích deklarativním způsobem

4 Použité prvky bezpečnosti a jejich nastavení

Pro účel zabezpečení systému, který bude vytvořen, budou použity kódové skupiny a sady oprávnění. Bude vytvořena kódová skupina, do které by měly být zařazeny všechny assembly přeložené a spuštěné tímto systémem. Tato skupina bude mít přidělenou sadu oprávnění vytvořenou pro tento účel. Tato sada oprávnění by měla obsahovat dostatečná oprávnění, která povolují studentům provést vše, co je po nich žádáno v rámci výuky, a zároveň jim nedovolit nic jiného, co by mohlo ohrozit stabilitu systému.

XML soubor obsahující sadu oprávnění vytvořenou pro tento systém je znázorněn ve výpisu kódu 16. Tato sada obsahuje oprávnění pro přístup k proměnné prostředí *TEMP*, zápis do adresáře *TEMP*, reflexi, povolení spuštění assembly, přístup k DNS, soketům, webu a SQL klientům. Naopak jim nebude povolen přístup do registrů, manipulace se zásadami zabezpečení, přístup k protokolu událostí a tiskárnám. Také na serveru nebude možné spouštět desktopové aplikace, jejichž testování pomocí spuštění není možné.

```
<PermissionSet class="NamedPermissionSet"
      version="1"
      Name="Maus"
      Description="Toto je sada oprávnění vytvořená pro systém MAUS, oprávnění jsou
          přidělena všem programům spouštěným tímto systémem.">
    <IPermission class="EnvironmentPermission"
      version="1"
      Read="TEMP" />
    <IPermission class="FileDialogPermission"
      version="1" />
    <IPermission class="ReflectionPermission"
      version="1"
      Unrestricted="true" />
    <IPermission class="RegistryPermission"
      version="1" />
    <IPermission class="SecurityPermission"
      version="1"
      Flags="Execution" />
    <IPermission class="UIPermission"
      version="1"
      Clipboard="OwnClipboard" />
    <IPermission class="DnsPermission"
      version="1"
      Unrestricted="true" />
    <IPermission class="PrintingPermission"
      version="1"
      Level="NoPrinting" />
    <IPermission class="EventLogPermission"
      version="1" />
    <IPermission class="SocketPermission"
      version="1"
      Unrestricted="true" />
    <IPermission class="WebPermission"
      version="1"
      Unrestricted="true" />
    <IPermission class="DirectoryServicesPermission"
```

```
        version="1">
            <Path name="%TEMP%"
         access="Write" />
      </IPermission>
      <IPermission class="SqlClientPermission"
         version="1"
         AllowBlankPassword="False" />
</PermissionSet>
```

Výpis 16: Sada oprávnění pro MAUS

Kódová skupina obsahuje výše uvedenou sadu oprávnění. Tato skupina má přidělen atribut *LevelFinal*, aby nebylo možné z této skupiny dále dědit. Také má přidělen atribut *Exclusive*, což označuje, že assembly, které budou zařazeny do této skupiny, budou mít pouze oprávnění přidělená této skupině. Zabraňuje to přidělení dalších oprávnění, pokud by assembly byla zařazena i do další skupiny např. *MyComputer*, kdy by pak dostala všechna oprávnění. Podmínkou členství pro tuto skupinu je určena URL, ze které pochází. V tomto případě se jedná o umístění v adresáři *c:\temp\maus*, odkud budou spouštěny programy studentů. XML soubor pro tuto kódovou skupinu je ve výpis kódu 17.

```
<CodeGroup class="UnionCodeGroup"
      version="1"
      PermissionSetName="Maus"
      Attributes="Exclusive, LevelFinal"
      Name="Maus_Group"
      Description="Tato zóna odpovídá programům spouštěným v systému MAUS, tyto
          programy by měly mít omezený přístup k systému">
   <IMembershipCondition class="UrlMembershipCondition"
         version="1"
         Url="file://C:/temp/maus/*" />
</CodeGroup>
```

Výpis 17: Kódová skupina pro MAUS

5 Možnosti testování studentských projektů a nástroje pro testování

Tato kapitola se zabývá možnostmi automatizovaného testování studentských projektů, které pak budou využity pro implementaci systému. Dále se pak zabývá testovacími nástroji použitelnými na platformě .NET.

5.1 Automatizované testování studentských projektů

Způsoby testování studentských projektů:

1. Unit testy - lze provádět, pokud je známa struktura aplikace. V tomto případě je potřeba mít unit test a ten se pak spouští. Na základě počtu testovacích metod, které projdou, se dá vyhodnotit počet bodů. Rovněž je možné ohodnotit jednotlivé testovací metody počtem bodů, které pak budou přiděleny, pokud tyto testovací metody úspěšně projdou. Zadáním těchto úkolů pak bude, buď částečně hotový kód, kde student doplní těla metod, nebo specifikace struktury kódu, která odpovídá očekávané struktuře pro testování.

2. Vstupy a výstupy - je možné provádět pro konzolové aplikace, kde je nutné mít soubor(y) se vstupními parametry a soubor(y) s očekávaným výstupem. Hodnotit se pak dají podle počtu řádků, které se ve výstupním souboru shodují, nebo podle počtu výstupů, které se shodují v případě, že by pro jeden úkol byl víc než jeden vstupní a výstupní soubor, a tyto různé soubory testovaly různé vlastnosti aplikace. Výhodnější je ale více různých souborů s různými vlastnostmi, poněvadž je to snadnější pro implementaci. Zadáním těchto úkolů pak bude konzolová aplikace, u které je jasné, jaké výstupy dostanu ke konkrétním vstupům.

3. Překlad - dá se testovat u všech aplikací. Pro překlad je nutné mít k dispozici všechny zdrojové soubory a knihovny, které aplikace používá a nejsou součástí platformy, na které se testuje (.NET, Java API). Jenom na tomto výsledku však nemůže být založeno hodnocení, jelikož výstupem takového testování je pouze úspěšně, či neúspěšně přeloženo.

5.2 Nástroje pro testování v .NET

Mezi nejpoužívanější nástroje pro unit testování patří NUnit a Visual Studio Team Test, kterými se budou zabývat následující kapitoly. O porovnání těchto nástrojů se můžete dočíst v [19].

5.2.1 NUnit

NUnit je framework pro unit testování na platformě .NET. Umožňuje testování pro všechny jazyky platformy .NET. NUnit byl původně vytvořen portací z JUnit a později přepracován a přizpůsoben platformě .NET. Aktuální verze je 2,4 a je napsána v jazyce

C#. Pro testování je možné využít samostatnou desktopovou aplikaci nebo příkazový řádek.

Ukázku testu napsaného v NUnit můžete vidět ve výpisu kódu 18, který byl převzat z [19]. Pro vytvoření testu se používají atributy, které se píšou na úrovni tříd a metod. Mezi nejdůležitější atributy patří *[TestFixture]*, který označuje testovací třídu, a *[Test]* který označuje testovací metodu. Mezi další atributy pak patří *[SetUp]* a *[TearDown]*, které připravují proměnné pro testování a uklízejí po testu. Atribut *[ExpectedException(„mojeVyjimka“)]* určuje, že očekávaným chováním testovací metody bude vyhození výjimky. Atribut *[Ignore(„Duvod“)]* říká, že metoda bude při spuštění testu ignorována.

Pro vyhodnocení, zda testovací metoda uspěla nebo ne, se používají např. metody *Assert.AreEqual(„očekávaná hodnota“, „aktuální hodnota“)* nebo *Assert.IsTrue(„hodnota“)* a další.

```
using System;
using NUnit.Framework;

[TestFixture]
public class AccountTest
{
    Account source;
    Account destination;

    [SetUp]
    public void Init ()
    {
        source = new Account();
        source.Deposit(200f);
        destination = new Account();
        destination.Deposit(150f);
    }

    [Test]
    public void TransferFunds()
    {
        source.TransferFunds(destination, 100f);
        Assert.AreEqual(250, destination.Balance);
        Assert.AreEqual(100, source.Balance);
    }

    [Test]
    [ExpectedException(typeof(InsufficientFundsException))]
    public void TransferWithInsufficientFunds()
    {
        source.TransferFunds(destination, 300f);
    }

    [Test]
    [Ignore("Decide how to implement transaction management")]
    public void TransferWithInsufficientFundsAtomicity()
    {
        try
        {
```

```
            source.TransferFunds(destination, 300f);
        }
        catch (InsufficientFundsException ex) {}
        Assert.AreEqual(200f, source.Balance);
        Assert.AreEqual(150f, destination.Balance);
    }
}
```

Výpis 18: Test napsaný v NUnit z [19]

5.2.2 Visual Studio Team Test

Team Test je framework pro unit testy integrovaný do Visual Studia. Tyto testy se vytváří a spouští přímo ve Visual Studiu. Team Test ovšem není dostupný ve všem verzích Microsoft Visual Studia.

Ukázku testu napsaného v Team Test můžete vidět ve výpisu kódu 19, která byla převzata z [19]. Tato ukázka odpovídá stejnému testu jako pro NUnit. Pro definování testovací třídy se používá atribut *[TestClass()]*. Pro označení testovací metody je použit atribut *[TestMethod()]*. Rovněž se dá připravit proměnné pro testování a uklidit po testu s pomocí atributů *[ClassInicialize()]* a *[ClassCleanup()]*. I tady existuje atribut, který určuje, že očekávaným chováním testovací metody bude vyhození výjimky, a to *[ExpectedException(„mojeVyjimka“)]*. Stejně tak atribut pro ignorování metody při spuštění testu je *[Ignore]*. Kromě těchto nejčastěji používaných atributů existuje i několik dalších, jejich seznam můžete najít v [19].

Podobně jako u NUnit testů se pro vyhodnocení, zda testovací metoda uspěla nebo ne, používají např. metody *Assert.AreEqual(„očekávaná hodnota“,„aktuální hodnota“)*, *Assert.IsTrue(„hodnota“)*, *Assert.IsNotNull(„hodnota“)* a další. Výčet těchto metod ovšem není úplně stejný s NUnit a není ani tak široký.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;

[TestClass()]
public class AccountTest
{
    private TestContext testContextInstance;
    private static Account source;
    private static Account destination;

    public TestContext TestContext
    {
        get { return testContextInstance; }
        set { testContextInstance = value; }
    }

    [ ClassInitialize () ]
    public static void Init (TestContext testContext)
    {
        source = new Account();
```

```
        source.Deposit(200f);
        destination = new Account();
        destination.Deposit(150f);
    }

    [TestMethod()]
    public void TransferFunds()
    {
        source.TransferFunds(destination, 100f);
        Assert.AreEqual(250f, destination.Balance);
        Assert.AreEqual(100f, source.Balance);
    }

    [TestMethod()]
    [ExpectedException(typeof(InsufficientFundsException))]
    public void TransferWithInsufficientFunds()
    {
        source.TransferFunds(destination, 300f);
    }

    [TestMethod()]
    [Ignore]
    public void TransferWithInsufficientFundsAtomicity()
    {
        try
        {
            source.TransferFunds(destination, 300f);
        }
        catch (InsufficientFundsException ex) { }
        Assert.AreEqual(200f, source.Balance);
        Assert.AreEqual(150f, destination.Balance);
    }
}
```

Výpis 19: Test napsaný ve Visual Studio Team Test z [19]

5.2.3 Porovnání nástrojů pro testování

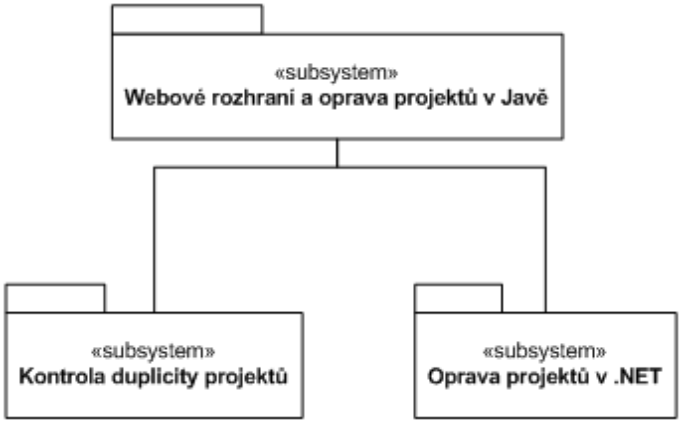
Jak se lze dočíst v předchozích kapitolách, vytváření testů v NUnit i Team Testu je velmi podobné. Výhodou NUnit je především to, že tento nástroj pro testování je zdarma a je to open source. NUnit je také považován za standart v unit testování. Naproti tomu Team Test poskytuje lepší uživatelské prostředí pro testování, protože spouštění testů je možné přímo ve Visual Studiu.

Nevýhodou Team Testu je ovšem to, že nemůže být spuštěn bez Visual Studia, což je pro použití v této diplomové práci zcela zásadní překážka. Testy by se totiž měly spouštět v rámci vytvořeného modulu, nikoliv z Visual Studia.

Vzhledem k tomu, že NUnit je zcela nezávislý a patří mezi nejčastěji používané nástroje pro unit testy, byl vybrán pro použití i v této práci.

6 Systém pro automatizované opravy projektů MAUS

MAUS je systém, který se zabývá opravou projektů v prostředí platforem .NET a Java a kontrolou duplicity projektů. Skládá se ze tří částí, které jsou vytvořeny v rámci tří diplomových prací. Jelikož jsou tyto části vytvořeny ve dvou různých platformách, kterými jsou Java a .NET, byly pro komunikaci mezi nimi zvoleny webové služby. Tyto tři části mají také společnou databázi, ve které jsou uloženy data o všech odevzdaných řešeních projektů a další potřebné informace. Strukturu těchto částí si můžete prohlédnout na obrázku 6.



Obrázek 6: Struktura systému MAUS

První z těchto částí se zabývá opravou projektů v jazyce Java s pomocí testů vytvořených v JUnit. Tato část rovněž obsahuje webové rozhraní pro komunikaci s uživateli, kterými jsou vyučující, kteří vkládají testy a dostávají výsledky testů, a studenti, kteří vkládají svá řešení projektů, která mají být opravena. Kromě vkládání projektů rovněž tato část poskytuje možnost klasických testů, kdy studenti pouze odpovídají na otázky. Celá tato část je vytvořena v jazyce Java. Více o této části se můžete dozvědět z diplomové práce Bc. Radima Velčovského.

Druhou částí je kontrola duplicity projektů, která zjišťuje, zda více různých studentů neodevzdalo jako projekt tentýž kód. Tato část je vytvořena na platformě .NET v jazyce C# a více o ní se můžete dozvědět z diplomové práce Bc. Radka Stromského.

Třetí část se zabývá opravou projektů v jazyce C# na platformě .NET. Tato část je obsahem této diplomové práce, a proto se více o této části dozvíte v následujících kapitolách.

6.1 Specifikace požadavků

Tato kapitola se zabývá specifikací požadavků kladených na vývoj modulu pro opravu projektů v jazyce C# na platformě .NET. A dále pak funkčními a nefunkčními požadavky, které určují požadované funkce modulu.

6.1.1 Funkční požadavky

Cílem je vytvořit subsystém, který se bude zabývat opravou projektů na platformě .NET. Tento subsystém rozšiřuje podobný systém, který je vytvořen v Javě. Jelikož je dnes programovaní v jazyce C# poměrně rozšířené a vyučuje se jako předmět na škole, je potřeba vytvořit takový systém i pro tento jazyk. Tento subsystém by měl usnadnit práci vyučujících v opravě projektů, což museli až do teď dělat ručně. Tato práce je však zdlouhavá a časově náročná, obzvláště při velkém počtu studentů. Výhodou tohoto systému je také to, že student dostane své výsledky téměř okamžitě a bude mít čas na opravu svého řešení, pokud nebude zcela úspěšné.

Oprava projektů studentů by měla probíhat jedním ze dvou způsobů. Prvním z těchto způsobů je vytvoření unit testu, který bude spuštěn a na základě výsledků dostane student hodnocení. Druhým způsobem je vytvoření souborů obsahujících vstupy a k nim odpovídající výstupy pro jednotlivá řešení. Podle počtu výstupů, které odpovídají očekávaným a tudíž správným výstupům, pak bude student ohodnocen.

Vstupem tohoto subsystému jsou řešení projektů studenty, unit testy odpovídající jednotlivým zadáním a vstupní a výstupní soubory pro jednotlivé úlohy.

Výstupem je pak ohodnocení řešení studenta, které se skládá buď z metod unit testu, které proběhly úspěšně a neúspěšně, nebo ze seznamu vstupních souborů, se kterými běh studentova řešení proběhl správně a nesprávně, tzn. jejich výstupní soubor odpovídal očekávání, nebo ne.

Funkcemi modulu jsou překlad zdrojového kódu, jeho spuštění, spuštění unit testu a získání seznamu testovacích metod unit testu, který je potřebný pro ohodnocení těchto metod vyučujícím.

6.1.2 Nefunkční požadavky

Testovací jádro systému pro .NET bude implementován v jazyce C# na platformě .NET 3,5. Jako databázový server bude použit Microsoft SQL Server 2008. Komunikace s ostatními částmi systému bude probíhat pomocí webových služeb.

6.2 Analýza subsystému

Následující kapitola se zabývá analýzou testovacího jádra subsystému pro .NET, datové i aplikační části. Nejprve bude popsána analýza chování systému a po té také datová analýza.

6.2.1 Analýza aplikační části

Tato kapitola se zabývá popisem chování subsystému. V první části jsou formou diagramu případů užití znázorněny vlastnosti subsystému. V další části bude popsáno chování pomocí scénářů případů užití. V třetí části je analýza chování modulu zobrazena pomocí diagramu aktivit.

Obrázek 7: Diagram případů užití

6.2.1.1 Diagram případů užití Diagram případů užití zobrazený na obrázku 7 popisuje obecné vlastnosti testovacího jádra systému. Je to „Porovnání vstupů a výstupů“, „Vyhodnocení testem“ a „Získání seznamu testovacích metod“.

„Porovnání vstupů a výstupů“ se skládá ze „Spuštění porovnání vstupů a výstupů“, které zahrnuje „Přeložení zdrojového kódu“, „Spuštění přeloženého zdrojového kódu“ a „Porovnání výstupu s očekávaným“.

„Vyhodnocení testem“ se skládá ze „Spuštění vyhodnocení testu“, které se zahrnuje „Přeložení zdrojového kódu“ a „Přeložení zdrojového kódu testu“.

„Získání seznamu testovacích metod“ zahrnuje „Zjištění seznamu metod testu“.

6.2.1.2 Scénáře případů užití V následujícím textu je uvedeno několik nejdůležitějších scénářů případů užití pro tento subsystém systému MAUS. Zbývající scénáře případů užití pak lze najít v příloze C.

UC 4 - Spuštění porovnání vstupů a výstupů

Záměr: Vyhodnocení porovnání výstupu po běhu zdrojového kódu s očekávaným výstupem.

Rozsah: Subsystém MAUS
Úroveň: podfunkce
Primární aktor: Systém
Účastníci a zájmy: Systém požaduje porovnání výstupu získaného spuštěním zdrojového kódu a očekávaného výstupu.
Vstupní podmínka: Existují vstupní soubory, očekávané výstupní soubory a zdrojový kód k vyhodnocení.
Minimální záruky: Systém je informován o neúspěchu překladu zdrojového kódu nebo výsledcích porovnání.
Záruky úspěchu: Jsou předány výsledky porovnání výstupních souborů.
Spouštěč: Systém předal požadavek na provedení vyhodnocení pomocí spuštění zdrojového kódu.
Hlavní scénář:

1. Systém načte zabalený zdrojový kód z databáze podle Id.
2. Systém rozbalí zdrojový kód do Adresáře.
3. Systém přeloží zdrojový kód nacházející se v Adresáři.
4. Systém načte z databáze seznam vstupních a výstupních souborů, které odpovídají odpovědi studenta.

   Kroky 5.-6. se opakují podle počtu vstupních souborů.

5. Systém spustí přeložený zdrojový kód podle vstupního souboru.
6. Systém porovná výstupní soubor s očekávaným.
7. Systém vrátí výsledky vyhodnocení.

Rozšíření:
1,4a. Databáze není k dispozici.
    1,4a1. Systém vyhodí výjimku.
1b. Zabalený zdrojový kód nebyl nalezen.
    1b1. Systém vyhodí výjimku.
3a. Při překladu se objevily chyby
    3a1. Systém vrátí seznam chyb při překladu.
5a. Spuštění přeloženého zdrojového kódu se nedařilo.
    5a1. Systém vyhodí výjimku.
6a. Jeden ze souborů nebyl nalezen.
    6a1. Systém vyhodí výjimku.

UC 5 - Spuštění vyhodnocení testem

Záměr: Vyhodnocení zdrojového kódu za pomocí unit testu.
Rozsah: Subsystém MAUS
Úroveň: podfunkce
Primární aktor: Systém
Účastníci a zájmy: Systém požaduje spuštění testu a jeho vyhodnocení.
Vstupní podmínka: Existuje unit test a zdrojový kód k vyhodnocení.
Minimální záruky: Systém je informován o neúspěchu překladu zdrojového kódu nebo výsledcích testu.
Záruky úspěchu: Jsou předány výsledky porovnání výstupních souborů.
Spouštěč: Systém předal požadavek na spuštění testu.
Hlavní scénář:

1. Systém načte zabalený zdrojový kód z databáze podle Id.
2. Systém rozbalí zdrojový kód do Adresáře.
3. Systém přeloží zdrojový kód nacházející se v Adresáři.
4. Systém načte z databáze zabalený zdrojový kód testu odpovídající odpovědi studenta.
5. Systém rozbalí zdrojový kód testu do Adresáře2.
6. Systém přeloží zdrojový kód testu s pomocí zdrojového kódu.
7. Systém spustí test.
8. Systém vyhodnotí výsledky testu.
9. Systém vrátí výsledky vyhodnocení.

Rozšíření:
1,4a. Databáze není k dispozici.
    1,4a1. Systém vyhodí výjimku.
1b. Zabalený zdrojový kód nebyl nalezen.
    1b1. Systém vyhodí výjimku.
3a. Při překladu se objevily chyby.
    3a1. Systém vrátí seznam chyb při překladu.
4b. Zabalený zdrojový kód testu nebyl nalezen.
    4b1. Systém vyhodí výjimku.
6a. Nepodařilo se přeložit zdrojový kód testu.
    6a1. Systém vyhodí výjimku.

UC 6 - Zjištění seznamu metod testu

Záměr: Získání seznamu metod unit testu.
Rozsah: Subsystém MAUS
Úroveň: podfunkce
Primární aktor: Systém
Účastníci a zájmy: Systém požaduje získání seznamu metod testu pro pozdější ohodnocení testu.
Vstupní podmínka: V databázi je uložen nový test.
Minimální záruky: Systém je informován o neúspěchu překladu zdrojového kódu testu nebo seznamu metod, které obsahuje.
Záruky úspěchu: Je předán seznam metod unit testu.
Spouštěč: Systém předal požadavek získání metod testu.
Hlavní scénář:

1. Systém načte z databáze zabalený zdrojový kód testu s daným Id.
2. Systém rozbalí zdrojový kód testu do Adresáře.
3. Systém přeloží zdrojový kód testu.
4. Systém pomocí reflexe načte seznam metod přeloženého zdrojového kódu.
5. Systém vrátí načtený seznam metod testu.

Rozšíření:
1a. Databáze není k dispozici.
    1a1. Systém vyhodí výjimku.
1b. Zabalený zdrojový kód testu nebyl nalezen.
    1b1. Systém vyhodí výjimku.
3a. Při překladu se objevily chyby.
    3a1. Systém vrátí seznam chyb při překladu.

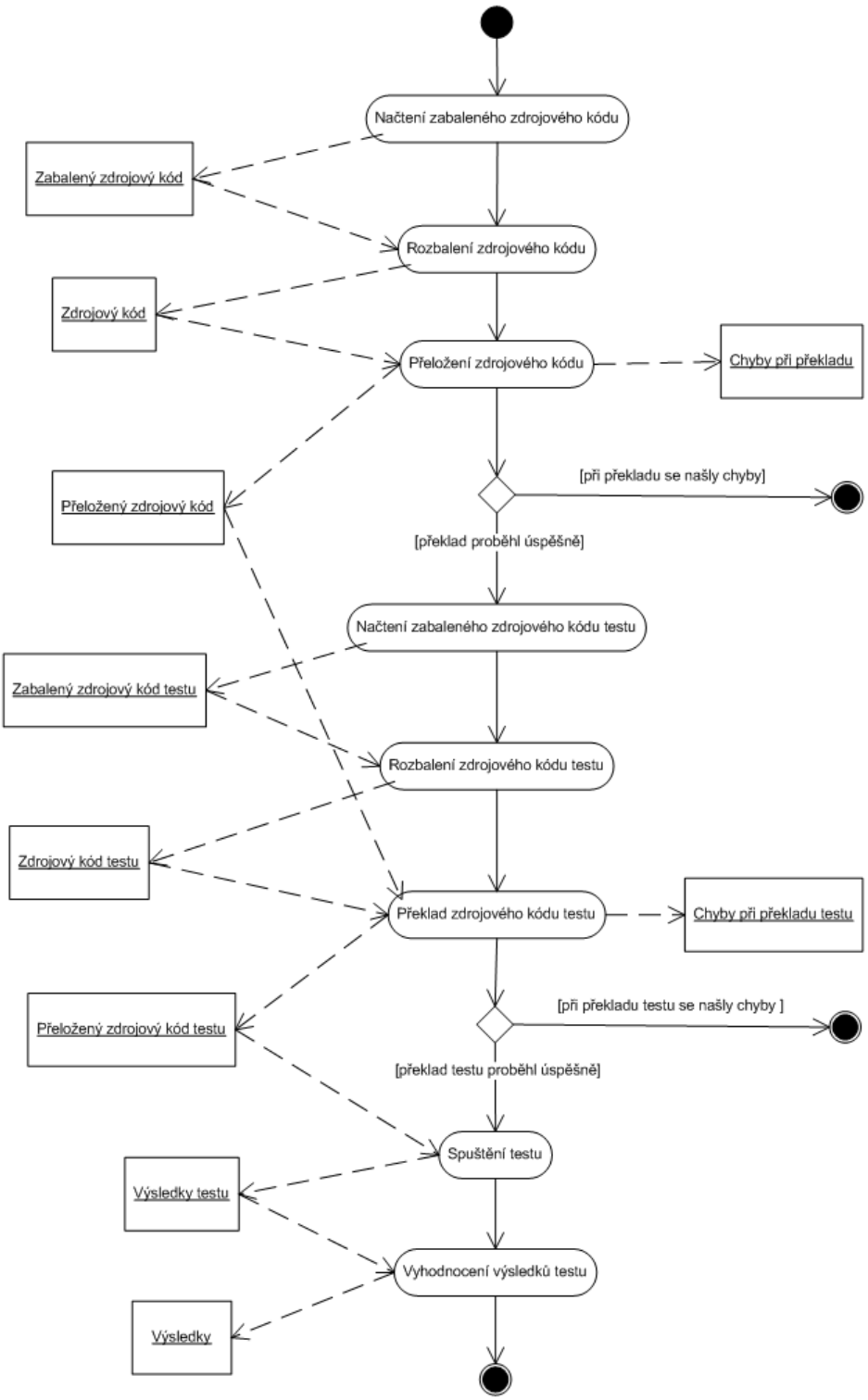
6.2.1.3 Diagram aktivit

Tato kapitola ukazuje několik základních diagramů aktivit.

Na obrázku 8 je znázorněn průběh automatického testu, kdy se porovnávají vstupy a výstupy. Nejdříve se načte zabalený zdrojový kód, který má být otestován, rozbalí se a přeloží. Pokud překlad proběhl v pořádku, je tento kód spuštěn s pomocí vstupních souborů, čímž vzniknou výstupní soubory, které jsou následně porovnány s očekávanými výstupy. Existuje-li další vstupní soubor, dochází k dalšímu spuštění, jinak dojde k vyhodnocení výsledku na základě porovnání výstupních souborů s očekávanými. Pokud překlad neproběhl úspěšně, pak tato aktivita končí.

Spuštění unit testů je zobrazeno na obrázku 9. Nejdříve se načte zabalený zdrojový kód, který má být otestován. Po té se rozbalí a přeloží. Pokud překlad proběhl v pořádku, pak se načtou a rozbalí zdrojové kódy testu. Do překladu testu se zahrnou i přeložené zdrojové kódy k otestování. Po úspěšném přeložení testů, dojde k jejich spuštění a následnému vyhodnocení výsledku. Pokud některý z překladů nebyl úspěšný, pak tato aktivita končí.

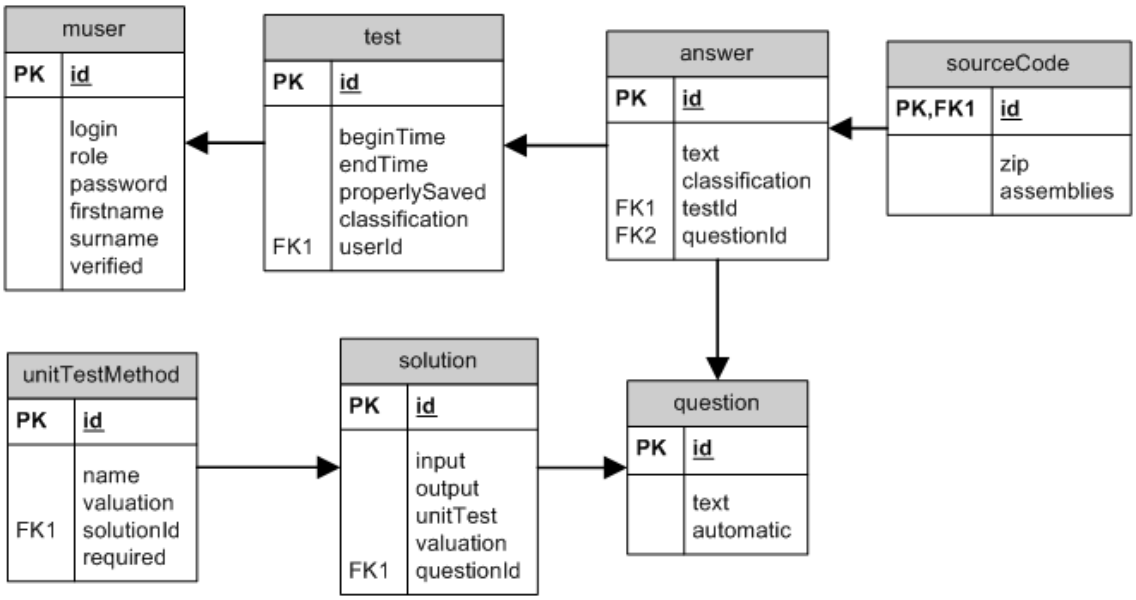
Obrázek 8: Diagram aktivit - Porovnání vstupu a výstupu



Obrázek 9: Diagram aktivit - Spuštění testů

6.2.2 Datová analýza

Datová analýza popisuje databázovou vrstvu systému, která je společná pro všechny části systému. ER diagram na obrázku 10 znázorňuje schéma části databáze, která je potřebná pro tento modul. ER diagram celé databáze se nachází v příloze B. Další informace ohledně databáze se můžete dozvědět v diplomové práci Bc. Radima Velčovského.



Obrázek 10: Schéma databáze

6.3 Návrh a implementace modulu

Kapitola návrh a implementace popisuje, jakým způsobem byl tento modul navržen a implementován a jaké technologie k tomu byly využity.

6.3.1 Použité technologie

V této kapitole se nachází stručný popis jednotlivých použitých technologií pro implementaci modulu.

6.3.1.1 ADO.NET Entity Data Model ADO.NET Entity Data Model je komponenta .NET Framework verze 3,5. Tato komponenta představuje objektově-relační mapování, kdy datový model relační databáze je namapován na objekty modelu v programovacím jazyce. Při běhu aplikace jsou pak požadavky napsané na objektech v programovacím jazyce překládány do SQL příkazů a posílány k provedení relační databázi. Výsledek dotazu vrácený relační databází je opět přeložen do objektů, se kterými lze pracovat v programovacím jazyce. Microsoft Visual Studio má pro toto objektově-relační mapování

plnou podporu a umožňuje tak zjednodušení práce programátorovi, který může využít grafické prostředí. Toto prostředí umožňuje vytvářet tabulky, atributy a vazby relačního modelu, ze kterých se pak vytvoří relační databáze, nebo naopak lze vygenerovat objekty z již existující relační databáze.

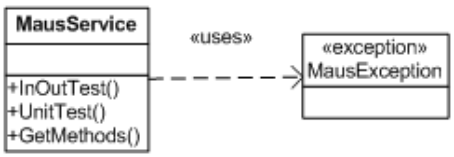
6.3.1.2 NUnit NUnit je nástroj pro testování za pomocí unit testu, více informací se nachází v kapitole 5.2.1.

6.3.2 Struktura aplikace

Aplikace je rozdělena do dvou projektů, kde každý z nich se stará o jinou část implementace. Těmito projekty jsou *WebService* a *TestCore*. Jejich vztah je vidět na obrázku 11.

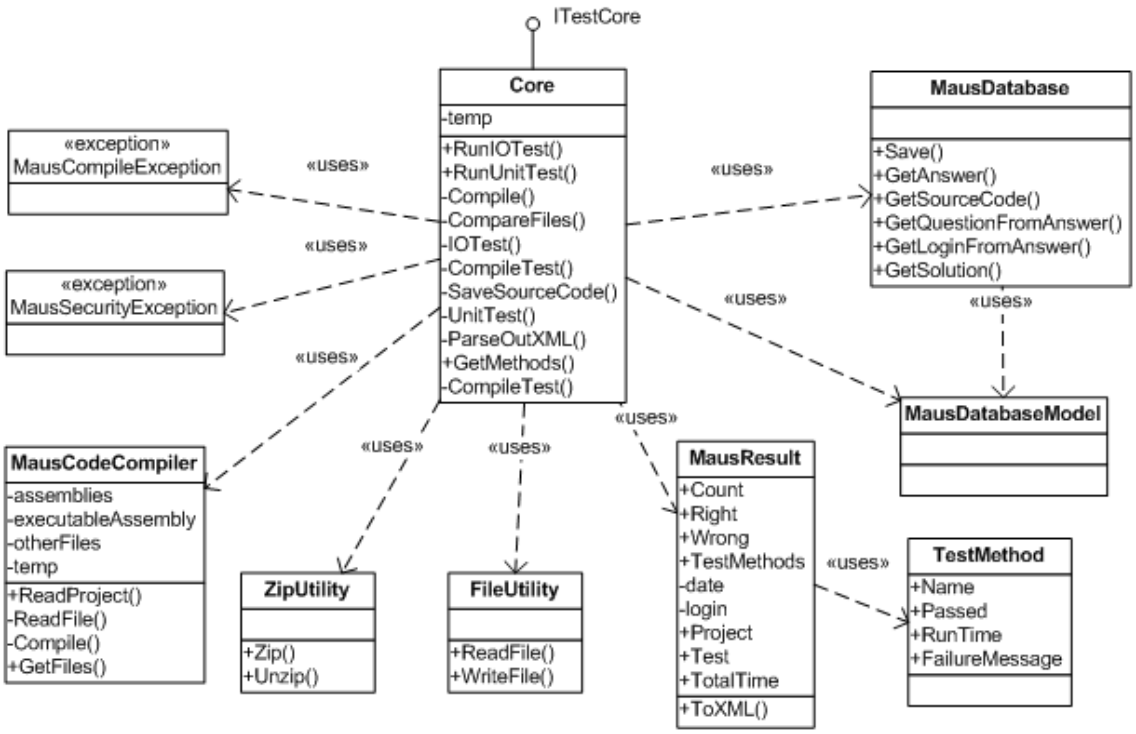
Obrázek 11: Schéma rozvržení implementace

Projekt *WebService* obsahuje webovou službu, která komunikuje s jinou částí systému, kterou je prezentační vrstva vytvořená v Javě. Poskytuje metody pro vyhodnocení testem, porovnání vstupů a výstupů a metodu pro získání seznamu testovacích metod, ke kterým mají být přiděleny body. Schéma projektu se nachází na obrázku 12. Třída *MausService* obsahuje uvedené webové služby. Třída *MausException* reprezentuje výjimku, která nastala v testovacím jádru.



Obrázek 12: Schéma projektu *WebService*

Projekt *TestCore* je implementací testovacího jádra. Jeho schéma je na obrázku 13. Rozhraní *ITestCore* poskytuje metody pro zjištění seznamu metod testu, spuštění vyhodnocení testem a spuštění porovnání vstupů a výstupů. Třída *Core* implementuje rozhraní *ITestCore*. Třída *MausCodeCompiler* se stará o překlad zdrojových kódů. Třída *MausCompileException* reprezentuje chyby, které nastaly při kompilaci zdrojových kódů. Třída *MausSecurityException* reprezentuje porušení zásad zabezpečení. Třída *ZipUtility* se zabývá komprimací a dekomprimací souborů. Třída *FileUtility* ukládá a načítá soubory z disku. Třída *MausResult* nese výsledek testování a obsahuje metodu pro generování výsledku, který je předáván webovou službou prezentační vrstvě. Struktura *TestMethod* obsahuje



Obrázek 13: Schéma projektu *TestCore*

informace o testované metodě. *MausDatabaseModel* je ADO.NET Entity Data Model, reprezentuje model databáze. Třída *MausDatabase* slouží pro načítání dat z databáze pomocí jejího modelu (*MausDatabaseModel*).

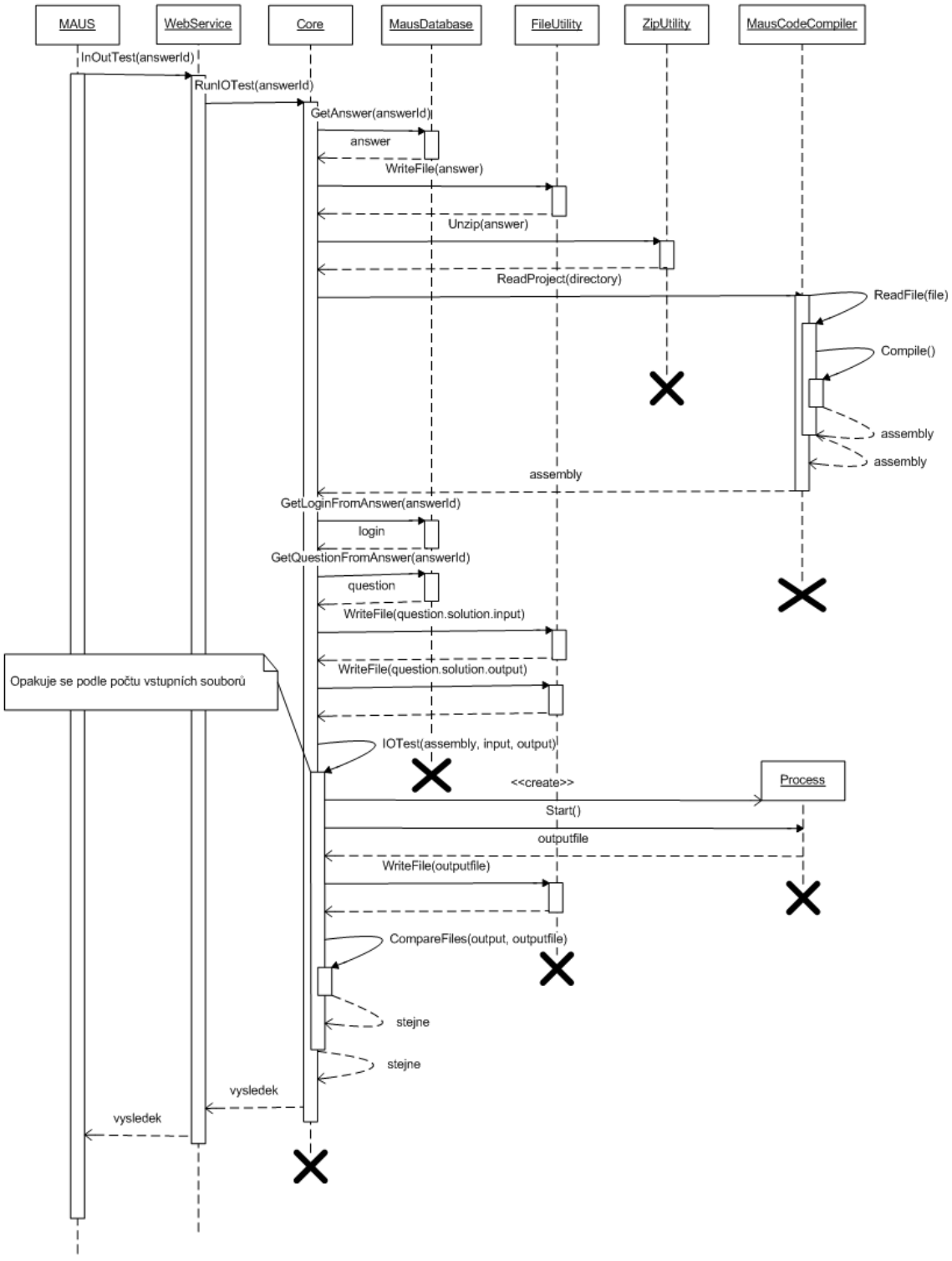
6.3.3 Implementace

Tato kapitola se zabývá implementací některých klíčových operací.

Průběh volání metody pro porovnání vstupů a výstupů lze vidět na obrázku 14. Podobným způsobem probíhají i ostatní metody.

Pro spouštění zdrojového kódu při porovnání vstupů a výstupů byl použit proces. Vlákna měla problém s pamětí a hůř se jim nastavovalo přesměrování vstupu a výstupu a také se hůř řídily, byť byly rychlejší. Způsob spouštění assembly lze vidět ve výpisu kódu 20.

```
/// <summary>
/// Spusti  aplikaci  se vstupy a porovna s ocekavanym vystupem
/// </summary>
/// <param name="assembly">assembly, ktera ma byt spustena</param>
/// <param name="input">cesta k souboru, ktera obsahuje vstupy</param>
/// <param name="output">cesta k souboru s ocekavanymi vystupy</param>
/// <returns>true pokud vystup aplikace odpovida ocekavanemu vystupu</returns>
private bool IOTest(Assembly assembly, string input, string output) {
```



Obrázek 14: Sekvenční diagram pro porovnání vstupů a výstupů

```
        string outputfile = input.Substring(0, input.LastIndexOf('_')) + "_output.txt";
        StringWriter sw = new StringWriter();
        FileStream fs = new FileStream(input, FileMode.Open);
        StreamReader sr = new StreamReader(fs);
        Process pr = new Process();
        pr.StartInfo.FileName = assembly.Location;
        pr.StartInfo.RedirectStandardInput = true;
        pr.StartInfo.RedirectStandardOutput = true;
        pr.StartInfo.RedirectStandardError = true;
        pr.StartInfo.UseShellExecute = false;
        pr.StartInfo.ErrorDialog = false;
        pr.Start();
        pr.StandardInput.Write(sr.ReadToEnd());
        pr.WaitForExit((int)TimeSpan.FromMinutes(limit).TotalMilliseconds);
        if (!pr.HasExited) {
            pr.Kill();
            throw new MausSecurityException("Beh procesu musel byt ukoncen. Bud doslo ke
                smycce, nebo byly poruseny zasady bezpecnosti.");
        }
        sw.Write(pr.StandardOutput.ReadToEnd());
        using (FileStream f = new FileStream(outputfile, FileMode.Create, FileAccess.Write)) {
            using (StreamWriter s = new StreamWriter(f)) {
                s.Write(sw.GetStringBuilder().ToString());
            }
        }
        return CompareFiles(outputfile, output);
    }
```

Výpis 20: Spuštění zdrojového kódu v procesu

V dalším výpisu kódu 21 je ukázán způsob porovnání výstupního souboru s očekávaným výstupem. Porovnávání se provádí tak, že se kontroluje, zda výstupní soubor obsahuje vše, co je v očekávaném souboru. Tento způsob byl zvolen z toho důvodu, že při psaní kódu studenti často zapisují do výstupu instrukce k zadávání údajů z klávesnice a takový výstupní soubor by pak neodpovídal očekávanému výstupu, byť by implementované řešení bylo správné.

```
    /// <summary>
    /// Porovna dva soubory a to tak, ze ve vytvoreny hleda to, co je ve spravnem.
    /// </summary>
    /// <param name="vytvoreny">cesta k vytvorenemu souboru</param>
    /// <param name="spravny">cesta k spravnemu souboru</param>
    /// <returns>true pokud ve vytvorenem nalezl vse, co je ve spravnem</returns>
    private bool CompareFiles(string vytvoreny, string spravny) {
        using (StreamReader srVytvoreny = new StreamReader(vytvoreny)) {
            using (StreamReader srSpravny = new StreamReader(spravny)) {
                string lineSpravny = "";
                while ((lineSpravny = srSpravny.ReadLine()) != null) {
                    string lineVytvoreny = "";
                    while ((lineVytvoreny = srVytvoreny.ReadLine()) != null) {
                        if (lineVytvoreny.Contains(lineSpravny)) {
                            break;
                        }
```

```
                }
                if (srVytvoreny.EndOfStream && !srSpravny.EndOfStream) {
                    return false;
                }
            }
            if (srSpravny.EndOfStream) {
                return true;
            } else {
                return false;
            }
        }
    }
}
```

Výpis 21: Porovnání výstupního souboru s očekávaným výstupem

Důležitou částí tohoto modulu je i překlad zdrojového kódu. Metoda, která má tento překlad na starosti, se nachází ve výpisu 22. Nejprve se nastaví parametry pro překlad, které byly získány již dříve, a pak se provede samotný překlad. Nakonec se zkontroluje, zda byl překlad proveden bez chyb. Pokud se objevily chyby, tak je vygenerována výjimka se seznamem chyb.

```
/// <summary>
/// Prelozi zdrojovy kod
/// </summary>
/// <param name="name">assembly name</param>
/// <param name="reference">cesty k souborum potrebnym k prekladu</param>
/// <param name="output">typ vystupu (spustitelny, knihovna,...)</param>
/// <param name="sourceCode">cesty k souborum, ktere se budou prekladat</param>
private void Compile(string name, List<string> reference, string output, List<string>
    sourceCode) {
    Dictionary<string, string> providerOptions = new Dictionary<string, string>();
    providerOptions.Add("CompilerVersion", "v3.5");
    CodeDomProvider com = new CSharpCodeProvider(providerOptions);
    CompilerParameters par = new CompilerParameters(reference.ToArray<string>());
    switch (output) {
        case "Exe": {
            par.CompilerOptions = "/t:exe";
            par.GenerateExecutable = true;
            par.OutputAssembly = temp + name + ".exe";
            break;
        }
        case "WinExe": {
            par.CompilerOptions = "/t:winexe";
            par.GenerateExecutable = true;
            par.OutputAssembly = temp + name + ".exe";
            break;
        }
        case "Library": {
            par.CompilerOptions = "/t: library ";
            par.OutputAssembly = temp + name + ".dll";
            break;
        }
        case "Module": {
```

```
                par.CompilerOptions = "/t:module";
                par.OutputAssembly = temp + name + ".netmodule";
                break;
            }
        }
          par.TempFiles.KeepFiles = false;
        par.GenerateInMemory = false;
        par.IncludeDebugInformation=true;
        CompilerResults res = null;
        try {
             res = com.CompileAssemblyFromFile(par, sourceCode.ToArray<string>());
        } catch (FileNotFoundException fnfe) {
             throw new MausCompileException(string.Format("Soubor {0} nebyl nalezen!",fnfe.
                  FileName),fnfe);
        }
        if (res.Errors.HasErrors) {
             StringBuilder sb = new StringBuilder();
            sb.AppendLine("Chyby pri prekladu:");
            foreach (CompilerError er in res.Errors) {
                int index = er.FileName.LastIndexOf('\\') + 1;
                sb.AppendLine(string.Format("{0}: {1} v souboru {2} na radku {3}",
                er.ErrorNumber, er.ErrorText,
                er.FileName.Substring(index, er.FileName.Length - index),
                er.Line));
             }
            throw new MausCompileException(sb.ToString());
        }
        if (par.GenerateExecutable == true) executableAssembly= res.CompiledAssembly;
        assemlies.Add(res.CompiledAssembly);
        otherFiles.Add(res.PathToAssembly.Substring(0, res.PathToAssembly.Length - 3) + "pdb");
    }
```

Výpis 22: Překlad zdrojového kódu

Způsob načítání informací o projektu lze vidět ve výpisu kódu 23. Toto načítání probíhá ze souboru s příponou *.csproj*, ve kterém jsou tyto informace uloženy. K překladu je potřeba vědět název assembly, reference, na které se assembly odkazuje, výstupní typ assembly, seznam souborů, které se budou překládat, ostatní reference na projekty, které je třeba přeložit, a ostatní obsah, který je potřeba pro spuštění přeložené assembly.

```
    /// <summary>
    /// Precte soubor *.csproj a podle jeho obsahu prelozi projekt a jeho zavislosti .
    /// </summary>
    /// <param name="fi">soubor podle ktereho se ma prekladat</param>
    /// <param name="fis">seznam ostatnich souboru *.csproj, ktere mohou slouzit po preklad
        zavislosti</param>
    private void ReadFile(FileInfo fi, List<FileInfo> fis, string filePath) {
          string name = "";
          List<string> reference = new List<string>();
        string output = "";
        List<string> code = new List<string>();
        using (XmlTextReader reader = new XmlTextReader(fi.FullName)) {
             int index = fi.FullName.LastIndexOf('\\');
            string path = fi.FullName.Substring(0, index+1);
```

```
            while (reader.Read()) {
                if (reader.NodeType == XmlNodeType.Element) {
                    switch (reader.Name) {
                        case "AssemblyName": name = reader.ReadElementContentAsString(); break
                            ;
                        case "Reference": {
                            // vlozi referenci do seznamu a zkopiruje potrebne soubory
                            ReadReference(filePath, reference, reader, path);
                            break;
                        }
                        case "OutputType": output = reader.ReadElementContentAsString(); break;
                        case "Compile": code.Add(path + reader.GetAttribute("Include")); break;
                        case "ProjectReference": {
                            while (reader.Name != "Name") {
                                reader.Read();
                            }
                            string project = reader.ReadElementContentAsString();
                            reference.Add(temp + project + ".dll");
                            if (!assemlies.Any(a=>a.Location==temp + project + ".dll")) {
                                FileInfo f = fis.First(file => file.Name == project + ".csproj");
                                fis.Remove(f);
                                ReadFile(f, fis, "");
                            }
                            break;
                        }
                        case "Content": {
                            string s~= reader.GetAttribute("Include");
                            if (!File.Exists(s)) {
                                try {
                                    File.Copy(path + s, ".\\" + s);
                                    otherFiles.Add(".\\" + s);
                                } catch (DirectoryNotFoundException) {
                                    Directory.CreateDirectory(s.Substring(0,s.LastIndexOf('\\')));
                                    File.Copy(path + s, ".\\" + s);
                                    otherFiles.Add(".\\" + s);
                                }
                            }
                            break;
                        }
                    }
                }
            }
            Compile(name, reference, output, code);
        }
```

Výpis 23: Načtení informací pro překlad

Webová služba vrací pro metody vyhodnocení testem a porovnání vstupu a výstupu výsledek ve formě *string*, který obsahuje zápis XML souboru podobný jako ve výpisu kódu 24. Toto XML obsahuje informace o testu, kterými jsou datum a čas provedení testu, název projektu a testované třídy, login studenta, kterému patří řešení, počet testů a doba,

po kterou testy běžely. Dále pak obsahuje seznam metod a k nim příslušnou úspěšnost provedení testu. Strukturu tohoto XML navrhl Bc. Radim Velčovský.

```
<?xml version="1.0" encoding="UTF-8"?>
<testclasses date="27.4.2009 22:31:47" login="vym017" projectname="Palindroms">
  <testclass name="PalindromyTest" testcount="3" runtime="0.070">
    <testmethod name="PalindromyTest.Program.GetPalindromTest" passed="True" runtime="
        0.040" />
    <testmethod name="PalindromyTest.Program.IsPalindromTest" passed="True" runtime="0.000"
        />
    <testmethod name="PalindromyTest.Program.ReverseTest" passed="True" runtime="0.000" />
  </testclass>
</testclasses>
```

Výpis 24: XML soubor obsahující výsledek testu

Pro metodu získání testovacích metod vrací webová služba pole *string*, které obsahuje jednotlivé názvy testovacích metod.

6.4 Praktické zkušenosti

Pro vyzkoušení tohoto modulu byly využity starší projekty studentů z předmětu Programování v C#, které posloužily, jak pro otestování překladu, tak i pro otestování funkčnosti obou typů testů.

Při vkládání projektů do databáze byl úspěšně otestován překlad kódu. Kdy modul je schopen přeložit jakýkoliv řízený kód, který obsahuje všechny potřebné knihovny. Tyto knihovny by měly být součástí projektu. Mezi vkládanými projekty bylo i několik projektů vytvořených v Mono, tyto projekty byly také úspěšně přeloženy. Problém při překladu může nastat pouze tehdy, pokud projekt obsahuje nějakou chybu, chybí potřebná knihovna, nebo v případě, kdy projekt obsahuje neřízený kód. Neřízený kód je všeobecně považován za nebezpečný a tudíž nebude tímto modulem podporován.

Další testovanou vlastností modulu bylo porovnání vstupů a výstupů. Při těchto testech může nastat problém, pokud se v kódu vyskytne příkaz *Console.ReadKey()*, který očekává načtení znaku z klávesnice, zatímco při tomto testu se načítá ze souboru. Tento případ tak končí výjimkou. Ovšem to není až zas tak velký problém, protože místo tohoto příkazu lze využít příkaz *Console.ReadLine()*, který má podobný účinek a pro zdržení ukončení programu se používá častěji. Je tedy nutné pouze upozornit studenty, aby používali výhradně příkaz *Console.ReadLine()*.

Při vyhodnocení unit testem je zapotřebí, aby projekt měl požadovanou strukturu, jinak nastane problém již v době překladu testu, který s touto strukturou počítá. Špatná struktura projektu tedy skončí chybou při překladu.

Rovněž bylo otestováno navržené zabezpečení. Toto zabezpečení bylo testováno při porovnání vstupů a výstupů, které je nejkritičtější z hlediska zabezpečení, jelikož se zde spouští naprosto neznámý kód implementovaný nějakým studentem. Pro tento test byl vybrán jeden z nejtypičtějších případů, kdy dochází k porušení zásad zabezpečení. Tímto případem je, že studenti uvádějí pevnou cestu k souboru, a tak by se mohli pokoušet číst nebo zapisovat na disk do míst, kde k tomu nemají přístup. Zdrojový kód použitý

pro tento test lze vidět ve výpisu kódu 25. Tento kód se pokouší zapsat soubor přímo na disk C:\, ale oprávnění pro zápis je povoleno pouze do adresáře *TEMP*. Tento pokus, stejně jako jiné pokusy o porušení zabezpečení, skončí tak, že proces, ve kterém je tento kód spuštěn skončí bezpečnostní výjimkou. Tato bezpečnostní výjimka je zapsána do protokolu událostí a proces je po časovém limitu ukončen. V tomto případě je pak dále tato chyba delegována přes webovou službu uživatelskému rozhraní.

```
public class Class1 {
        public static  void Main(string[] args){
            using (FileStream fs = new FileStream(@"C:\soubor.txt", FileMode.Create)) {
                using (StreamWriter sw = new StreamWriter(fs)) {
                        sw.WriteLine("Byl jsem tady. Fantomas");
                    sw.Close();
                }
                fs.Close();
            }
    }
}
```

Výpis 25: Zdrojový kód porušující zásady zabezpečení pro systém MAUS

7 Závěr

V rámci této práce byl shrnut způsob zabezpečení platformy .NET a to hlavně z hlediska přístupu ke kódu.

Zároveň byl navržen a implementován modul pro spouštění dvou typů testů v jazyce C# a také způsob zabezpečení tohoto modulu. Tento modul byl zahrnut do podobného systému implementovaného v jazyce Java obsahujícího uživatelské rozhraní. Tyto části pak spolu komunikují za pomoci webových služeb.

Práce se dále věnovala porovnání dvou hlavních nástrojů pro unit testování, kterými jsou NUnit a Team Test. Pro účel této práce byl vybrán NUnit.

Tento modul byl prakticky otestován na starších projektech studentů z předmětu Programování v C# a bylo přitom objeveno několik problémů. Tyto problémy však nejsou nikterak závažné a lze je řešit upozorněním studentů. V budoucnu se dá tento modul rozšířit o testování projektů v dalších jazycích na platformě .NET.

Milada Píšová

8 Literatura

[1] Poole, Charlie, *NUnit*, http://www.nunit.org/index.php?p=docHome&r=2.4.8, 2008.

[2] Hunt, Andrew and Thomas, David *Pragmatic Unit Testing in C# with Nunit, The Pragmatic Programmers*, Raleigh, North Carolina, USA. ISBN: 0974514012

[3] Vondrák, Ivo, *Úvod do softwarového inženýrství*, Ostrava:VŠB , Technická Univerzita Ostrava, 2002

[4] Sceppa, David, *Programming Microsoft ADO.NET 2.0 Core Reference*, Redmond: Microsoft Press, 2006.

[5] Robinson, Simon, *Professional C#, Third Edition*, Indianapolis: Wiley Publishing Inc., 2004.

[6] Microsoft Corporation *.NET Framework Developer's Guide: Security in the .NET Framework*,
http://msdn.microsoft.com/en-us/library/fkytk30f(VS.80).aspx, 2006

[7] Sonu Chauhan *Code Access Security using C# in VS.NET 2005*,
http://www.c-sharpcorner.com/UploadFile/chauhan_sonu57/
CodeAccess102172006033850AM/CodeAccess1.aspx, 2006

[8] Puš, Petr *Poznáváme C# a Microsoft .NET , 75. díl - Bezpečnost*,
http://www.zive.cz/Clanky/Poznavame-C-a-Microsoft-NET–75-dil–Bezpecnost/sc-3-a-130928/default.aspx, 2006.

[9] Puš, Petr *Poznáváme C# a Microsoft .NET , 76. díl - CAS*,
http://www.zive.cz/Clanky/Poznavame-C-a-Microsoft-NET–76-dil–CAS/sc-3-a-131125/default.aspx, 2006.

[10] Puš, Petr *Poznáváme C# a Microsoft .NET 77. díl - Programové použití CAS*,
http://www.zive.cz/Clanky/Poznavame-C-a-Microsoft-NET-77-dil–Programove-pouziti-CAS/sc-3-a-131467/default.aspx, 2006.

[11] Puš, Petr *Poznáváme C# a Microsoft .NET 78. Díl - CAS II*,
http://www.zive.cz/Clanky/Poznavame-C-a-Microsoft-NET-78-Dil–CAS-II/sc-3-a-131704/default.aspx, 2006.

[12] Puš, Petr *Poznáváme C# a Microsoft .NET 79. díl - Bezpečnost založená na rolích*,
http://www.zive.cz/Clanky/Poznavame-C-a-Microsoft-NET-79-dil–Bezpecnost-zalozena-na-rolich/sc-3-a-132189/default.aspx, 2006.

[13] LaMacchia, Brian A. *Security Through the Lifetime of a Managed Process: Fitting It All Together*, http://www.developer.com/security/article.php/3296261

[14] Wootton, Benjamin *Code Access Security*,
http://www.c-sharpcorner.com/UploadFile/benjaminwootton/
CodeAccessSecurity11242005070858AM/CodeAccessSecurity.aspx, 2004

[15] Rais, Razi Bin *Exploring Security in .NET - Part I*,
http://www.codeproject.com/KB/security/ExploringSecurityNET.aspx

[16] Parmar, Chandrakant *Writing Secure Code using C#*,
http://www.c-sharpcorner.com/UploadFile/chandrakantpp/
SecureCodeusingCSharp11232005005357AM/SecureCodeusingCSharp.aspx, 2001

[17] Rausch, Chris *Understanding how assert effects security*,
http://www.vbdotnetheaven.com/UploadFile/crausch/
UnderstandingAssert04202005071142AM/UnderstandingAssert.aspx

[18] Piliptchouk, Denis *Java vs. .NET Security, Part 3: Code Protection and Code Access Security (CAS)*,
http://www.onjava.com/pub/a/onjava/2004/01/28/javavsdotnet.html?page=3

[19] Vertigo Software, Inc. *Comparing nUnit to Team System Unit Testing*,
http://blogs.vertigosoftware.com/teamsystem/archive/2006/02/03/
Comparing_nUnit_to_Team_System_Unit_Testing.aspx

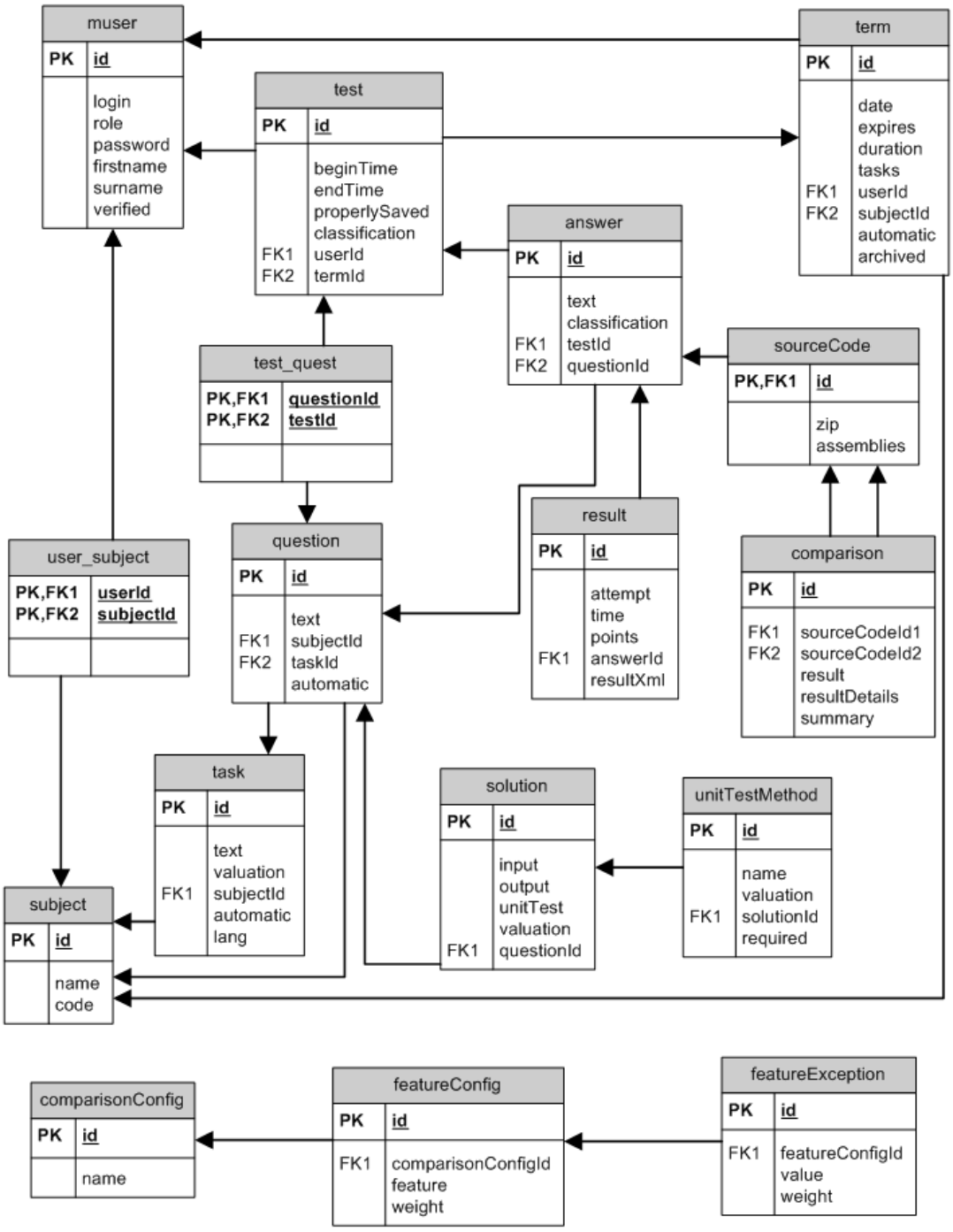
[20] Rausch, Chris *Viewing Assembly Permissions*,
http://www.c-sharpcorner.com/UploadFile/crausch/
ViewingAssemblyPermissions11232005045725AM/ViewingAssemblyPermissions.aspx

A Obsah CD

Adresář	Popis
/src	Zdrojové soubory modulu
/doc	Text diplomové práce

B Diagramy

Diagram uvedený v této sekci je úplným schématem databáze pro systém MAUS.



Obrázek 15: Schéma databáze systému MAUS

C Scénáře případů užití

UC 1 - Porovnání vstupů a výstupů

Záměr: Vyhodnocení odpovědi studenta za pomocí jeho spuštění a porovnání výstupů s očekávanými.

Rozsah: Systém Maus

Úroveň: podfunkce

Primární aktor: Maus

Účastníci a zájmy: Maus požaduje vyhodnocení odpovědi studenta

Vstupní podmínka: Existuje nevyhodnocená odpověď studenta.

Minimální záruky: Maus je vždy informován o vyhodnocení odpovědi studenta nebo chybě.

Záruky úspěchu: Jsou předány výsledky vyhodnocení.

Spouštěč: Maus předal požadavek na vyhodnocení odpovědi studenta.

Hlavní scénář:

1. Systém přijme Id odpovědi studenta, která má být vyhodnocena.
2. Systém spustí porovnání vstupů a výstupů a předá Id systému.
3. Systém předá výsledky Maus.

Rozšíření:

2a. Spuštění porovnání vstupů a výstupů selhalo.

2a1. Systém předá zprávu o chybě Maus

UC 2 - Vyhodnocení testem

Záměr: Vyhodnocení odpovědi studenta s pomocí unit testu.

Rozsah: Systém Maus

Úroveň: podfunkce

Primární aktor: Maus

Účastníci a zájmy: Maus požaduje vyhodnocení odpovědi studenta

Vstupní podmínka: Existuje nevyhodnocená odpověď studenta.

Minimální záruky: Maus je vždy informován o vyhodnocení odpovědi studenta nebo chybě.

Záruky úspěchu: Jsou předány výsledky vyhodnocení testem.

Spouštěč: Maus předal požadavek na vyhodnocení odpovědi studenta testem.

Hlavní scénář:

1. Systém přijme Id odpovědi studenta, která má být vyhodnocena testem.
2. Systém spustí vyhodnocení testu a předá Id systému.
3. Systém předá výsledky Maus.

Rozšíření:

2a. Spuštění testu selhalo.

2a1. Systém předá zprávu o chybě Maus

UC 3 - Získání seznamu testovacích metod

Záměr: Zjištění, které testovací metody obsahuje unit test.
Rozsah: Systém Maus
Úroveň: podfunkce
Primární aktor: Maus
Účastníci a zájmy: Maus požaduje zjištění, které testovací metody jsou obsaženy v testu pro jejich ohodnocení pro pozdější hodnocení výsledků testu.
Vstupní podmínka: Do databáze byl vložen nový test.
Minimální záruky: Maus je informován o chybě, nebo dostane seznam metod.
Záruky úspěchu: Je předán seznam testovacích metod unit testu.
Spouštěč: Maus předal požadavek na zjištění metod nového testu.
Hlavní scénář:

1. Systém přijme Id testu, u kterého je potřeba zjistit seznam metod.
2. Systém získá seznam metod testu a předá Id systému.
3. Systém předá seznam metod testu Maus.

Rozšíření:
2a. Získání seznamu metod selhalo.
2a1. Systém předá zprávu o chybě Maus.

UC 7 - Přeložení zdrojového kódu

Záměr: Přeložení zdrojového kódu.
Rozsah: Subsystém Maus
Úroveň: podfunkce
Primární aktor: Systém
Účastníci a zájmy: Systém požaduje přeložení zdrojového kódu.
Při neúspěchu překladu systém požaduje seznam chyb.
Vstupní podmínka: Zdrojový kód, který má být přeložen se nachází v Adresáři
Minimální záruky: Systém je vždy informován o úspěchu či neúspěchu překladu.
Záruky úspěchu: Zdrojový kód je přeložen.
Spouštěč: Systém požadoval překlad zdrojového kódu.
Hlavní scénář:

1. Systém načte z Adresáře soubory s informacemi o projektu.

   Kroky 2.-4. se opakují podle počtu souborů s informacemi o projektu.

2. Systém načte ze souboru obsahujícího informace o projektu seznam zdrojových souborů, seznam referencí a název assembly a typ assembly.
3. Systém nastaví parametry pro překlad za pomocí seznamu zdrojových kódů, seznamu referencí a názvu assembly a typ assembly.
4. Systém přeloží zdrojový kód podle parametrů nastavení.

5. Systém vrátí přeložený zdrojový kód.

Rozšíření:
2a. Reference je odkaz na nepřeložený kód.
    2a1. Provede se překlad nepřeloženého kódu.
    2a2. Scénář pokračuje krokem 3.
4a. Při překladu se vyskytly chyby.
    4a1. Systém vyhodí výjimku.

UC 8 - Spuštění přeloženého zdrojového kódu

Záměr: Spuštění přeloženého zdrojového kódu.
Rozsah: Subsystém Maus
Úroveň: podfunkce
Primární aktor: Systém
Účastníci a zájmy: Systém požaduje spuštění přeloženého zdrojového kódu.
Systém požaduje, pokud spuštěný zdrojový kód běží moc dlouho, jeho ukončení.
Systém požaduje přesměrování vstupů a výstupů z/do souboru.
Vstupní podmínka: Zdrojový kód je přeložen.
Existuje vstupní soubor.
Minimální záruky: Zdrojový kód je spuštěn.
Záruky úspěchu: Existuje výstupní soubor.
Spouštěč: Systém požadoval spuštění přeloženého zdrojového kódu.
Hlavní scénář:

1. Systém vytvoří proces z přeloženého zdrojového kódu.
2. Systém nastaví procesu jako vstup vstupní soubor.
3. Systém nastaví procesu výstup do paměti.
4. Systém spustí proces.
5. Systém počká určitý časový limit, až proces skončí.
6. Systém uloží výstup do souboru.

Rozšíření:
5a. Proces nebyl ukončen.
    5a1. Systém ukončí proces.
    5a2. Scénář pokračuje bodem 6.

UC 9 - Porovnání výstupu s očekávaným

Záměr: Porovnání výstupního souboru po spuštění zdrojového kódu s očekávaným výstupem.

Rozsah: Subsystém Maus
Úroveň: podfunkce
Primární aktor: Systém
Účastníci a zájmy: Systém požaduje porovnání dvou souborů.
Vstupní podmínka: Existuje výstupní soubor po spuštění zdrojového kódu.
Existuje očekávaný výstup programu.
Minimální záruky: Systém přečte očekávaný výstup.
Záruky úspěchu: Je předán výsledek porovnání.
Spouštěč: Systém požadoval porovnání souborů.
Hlavní scénář:

1. Systém otevře oba soubory pro čtení.

   Kroky 2.-4. se opakují dokud se nedosáhne konce souboru s očekávaným výstupem.

2. Systém přečte řádek ze souboru s očekávaným výstupem a uloží ho jako Spravny

   Kroky 3.-4. se opakují dokud se Spravny a Vytvoreny nerovnají, nebo čtení nedosáhne konce souboru.

3. Systém načte řádek ze souboru s výstupem pro spuštění zdrojového kódu a uloží ho jako Vytvoreny

4. Systém porovná Spravny a Vytvoreny

5. Systém vrátí výsledky porovnání.

UC 10 - Přeložení zdrojového kódu testu

Záměr: Přeložení zdrojového kódu.
Rozsah: Subsystém Maus
Úroveň: podfunkce
Primární aktor: Systém
Účastníci a zájmy: Systém požaduje přeložení zdrojového kódu testu.
Při neúspěchu překladu systém požaduje seznam chyb.
Vstupní podmínka: Zdrojový kód testu, který má být přeložen se nachází v Adresáři2
Minimální záruky: Systém je vždy informován o úspěchu či neúspěchu překladu.
Záruky úspěchu: Zdrojový kód testu je přeložen.
Spouštěč: Systém požadoval překlad zdrojového kódu testu.
Hlavní scénář:

1. Systém načte z Adresáře2 soubor s informací o projektu testu.

2. Systém načte ze souboru obsahujícího informace o projektu testu seznam zdrojových souborů, seznam referencí a název assembly a typ assembly.

3. Systém upraví referenci na testovanou assembly na aktuální zdrojový kód.
4. Systém nastaví parametry pro překlad za pomocí seznamu zdrojových kódů, seznamu referencí a názvu assembly a typ assembly.
5. Systém přeloží zdrojový kód testu podle parametrů nastavení.
6. Systém vrátí přeložený zdrojový kód testu.

Rozšíření:
5a. Při překladu se vyskytly chyby.
    5a1. Systém vyhodí výjimku.